

**МИНОБРНАУКИ РОССИИ**

**федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технологический университет «СТАНКИН»**

**(ФГБОУ ВО «МГТУ «СТАНКИН»)**

|  |  |
| --- | --- |
| **Институт**  **информационных**  **технологий** | **Кафедра**  **информационных технологий и вычислительных систем** |

ОТЧЕТ О ВЫПОЛНЕНИИ   
ИНДИВИДУАЛЬНОГО ЗАДАНИЯ ПО ДИСЦИПЛИНЕ   
«Методы оптимизации»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| СТУДЕНТА | *2* | КУРСА | | *бакалавриата* | ГРУППЫ | *ИДБ-23-16* |
|  | | | *(уровень профессионального образования)* | |  | |

|  |
| --- |
| **Грачева Николай Романовича** |
| *(ФИО)* |

НА ТЕМУ

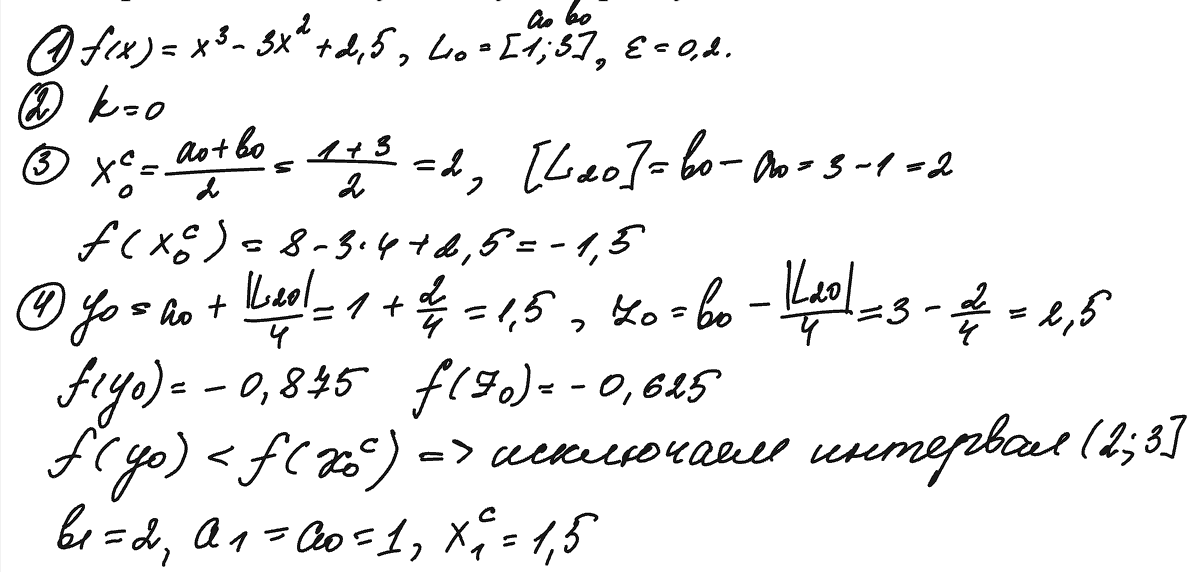
|  |
| --- |
| Лабораторная работа №1 «Одномерная оптимизация» Вариант № 8 |

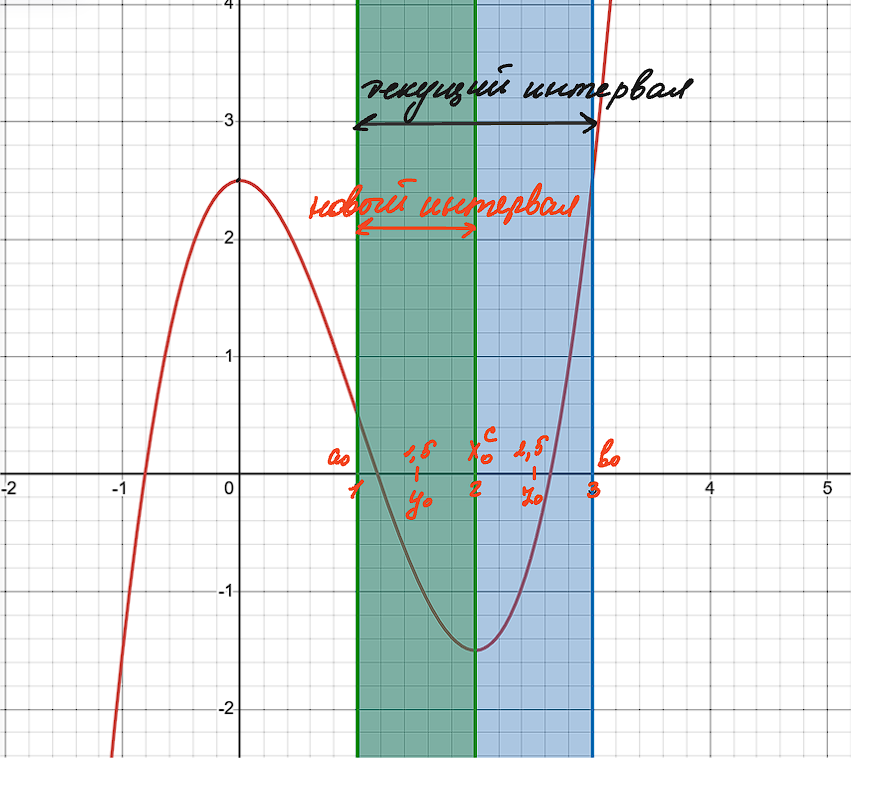
|  |  |
| --- | --- |
| Направление: | 09.03.04 Программная инженерия |
| Профиль подготовки: | Системный анализ и проектирование программных комплексов |

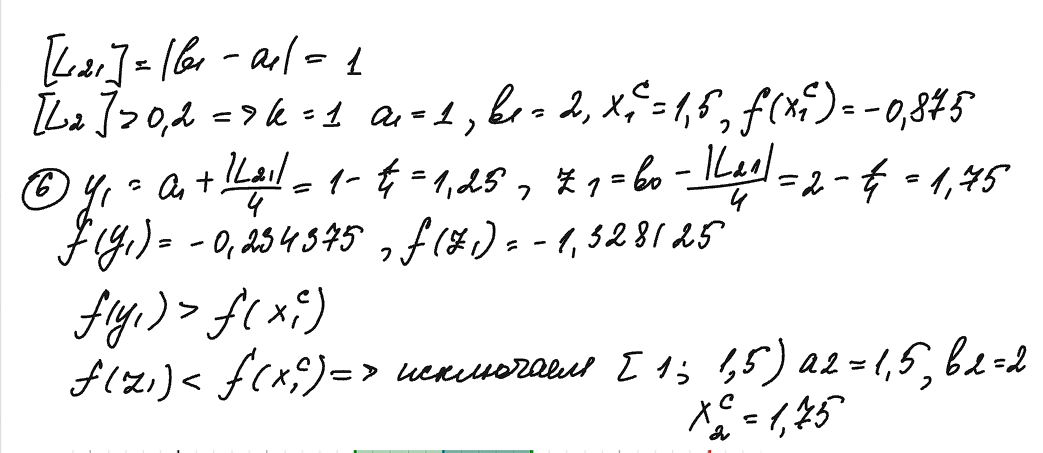
|  |  |  |  |
| --- | --- | --- | --- |
| Отчет сдан «\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г. | | | |
|  |  |  |  |
| Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | |
|  | | | |
| Преподаватель | Волкова О. Р., к.т.н. |  |  |
|  | *(Ф.И.О., должность, степень, звание.)* |  | *(подпись)* |

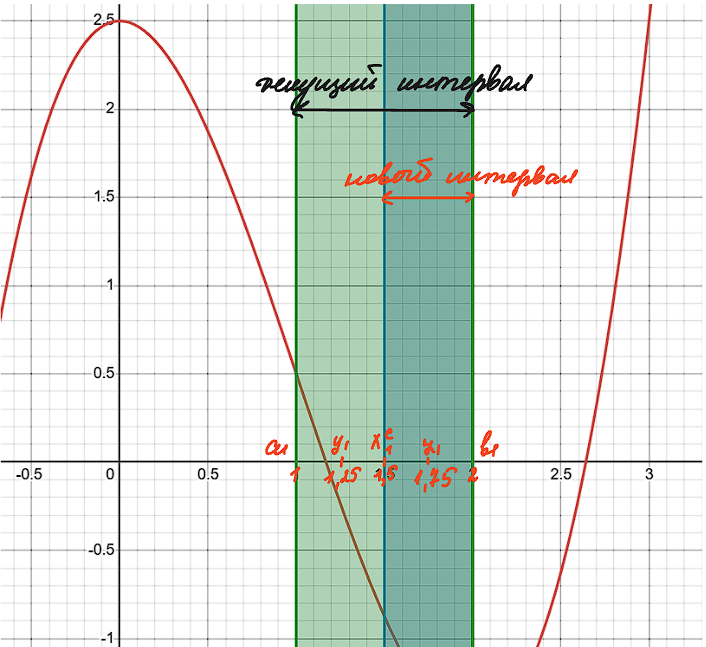
МОСКВА 2025

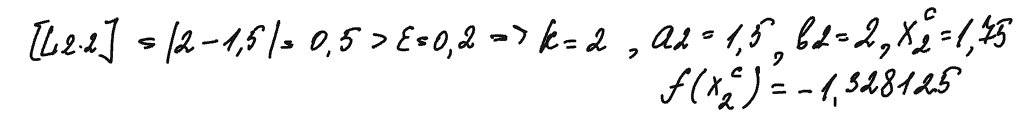
1. Найти минимум функции из индивидуального задания методом половинного деления. Проиллюстрировать на графике функции переход от начального интервала L0 к следующему интервалу L в соответствии с методом.

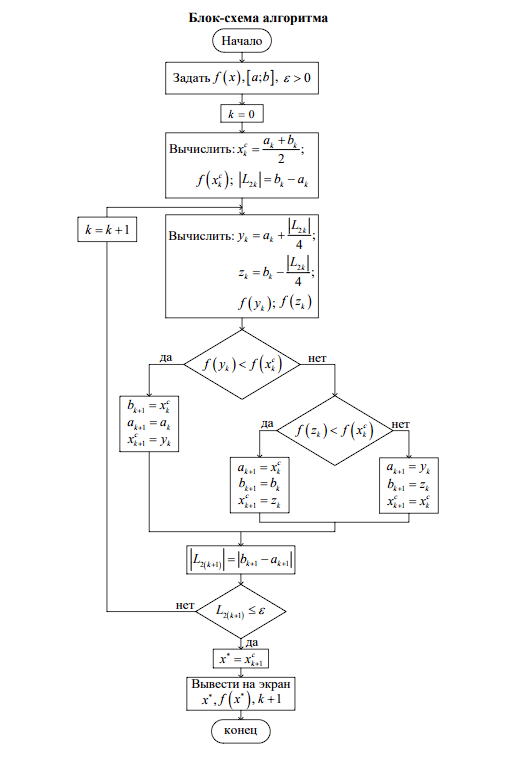




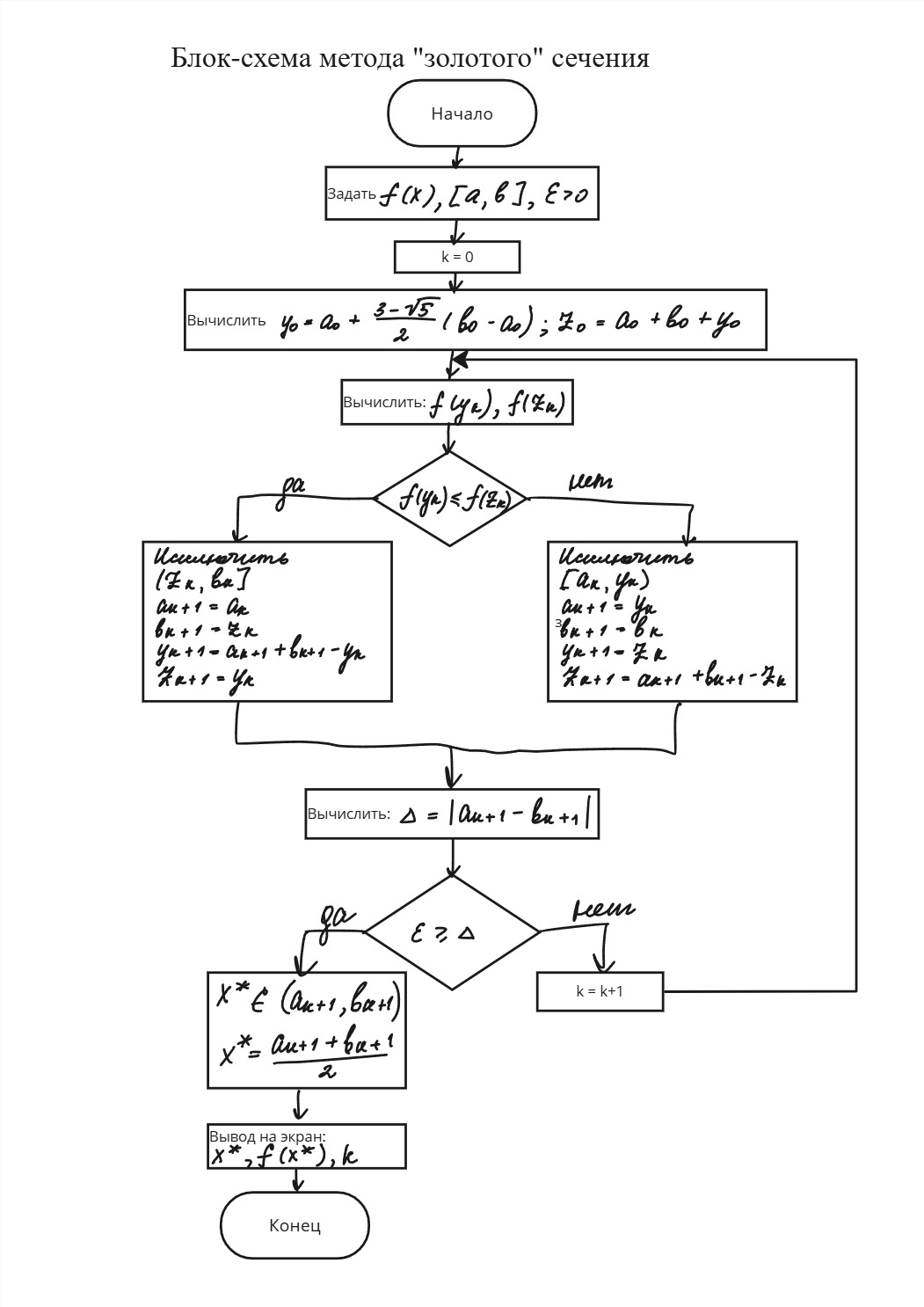




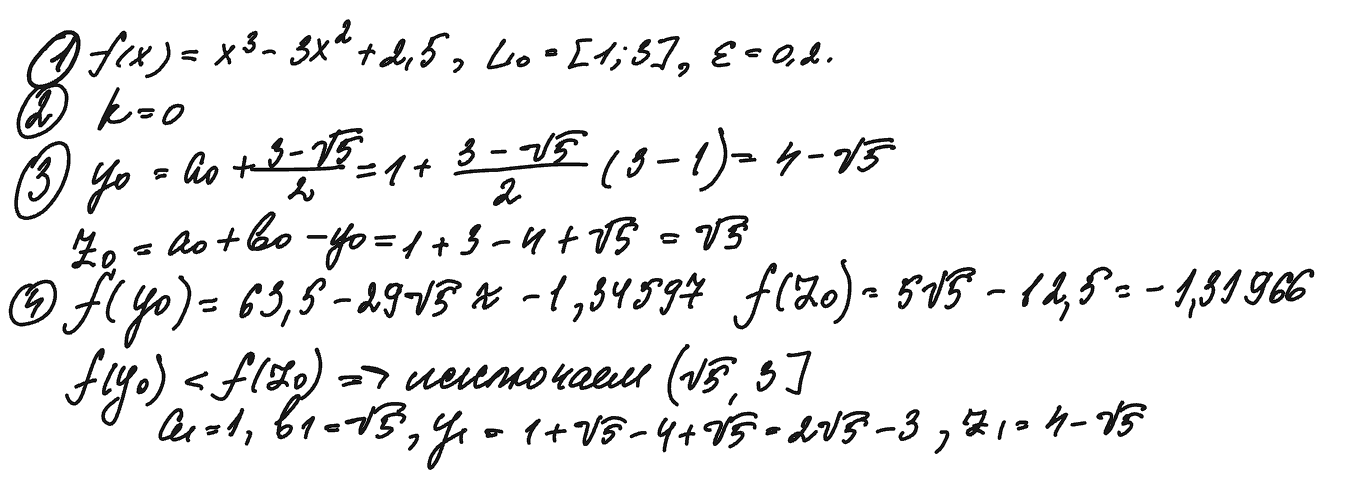


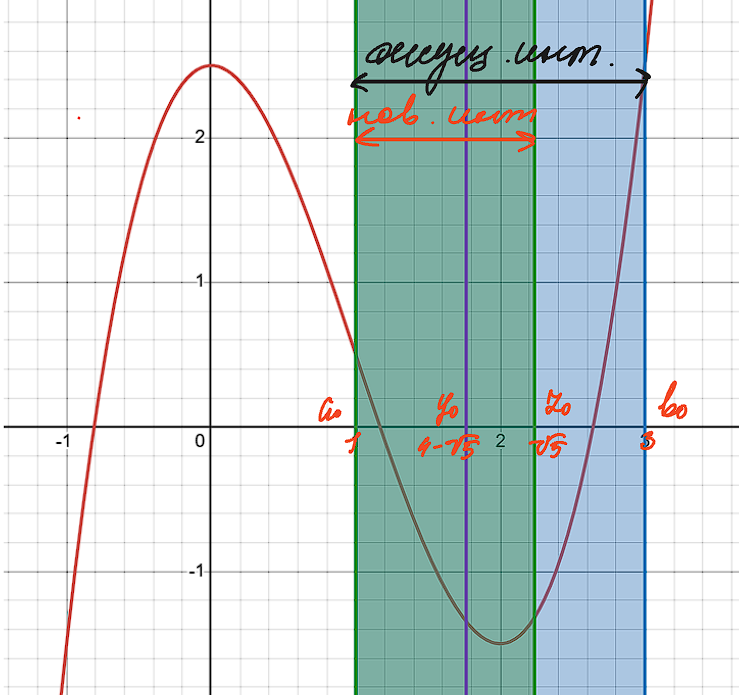


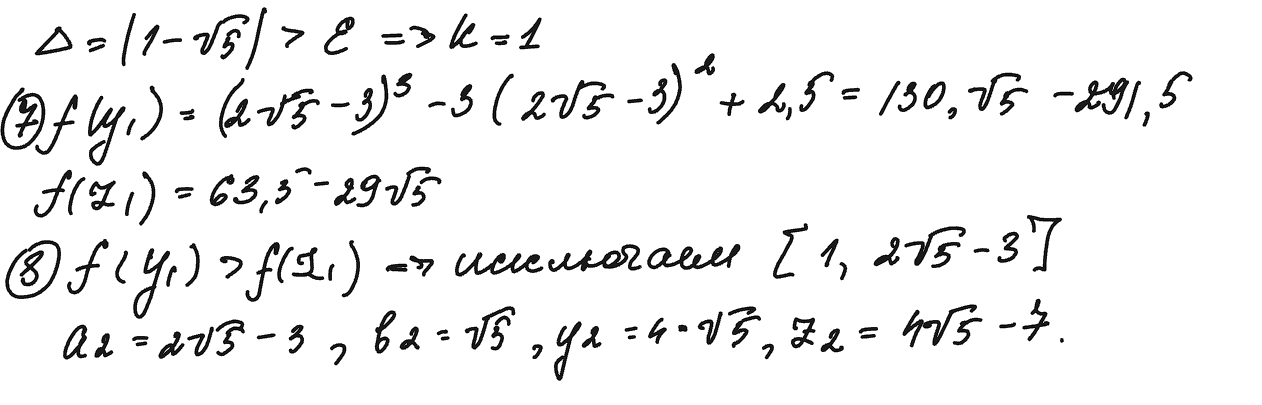
1. Построить блок-схему алгоритма поиска экстремума по методу «золотого» сечения.

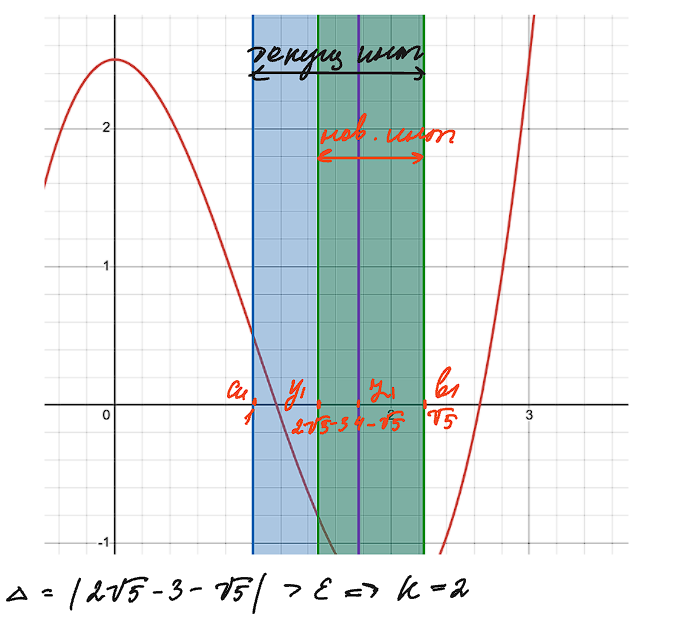


1. Найти минимум функции из индивидуального задания методом «золотого» сечения. Проиллюстрировать на графике функции переход от начального интервала L0 к следующему интервалу L в соответствии с методом.

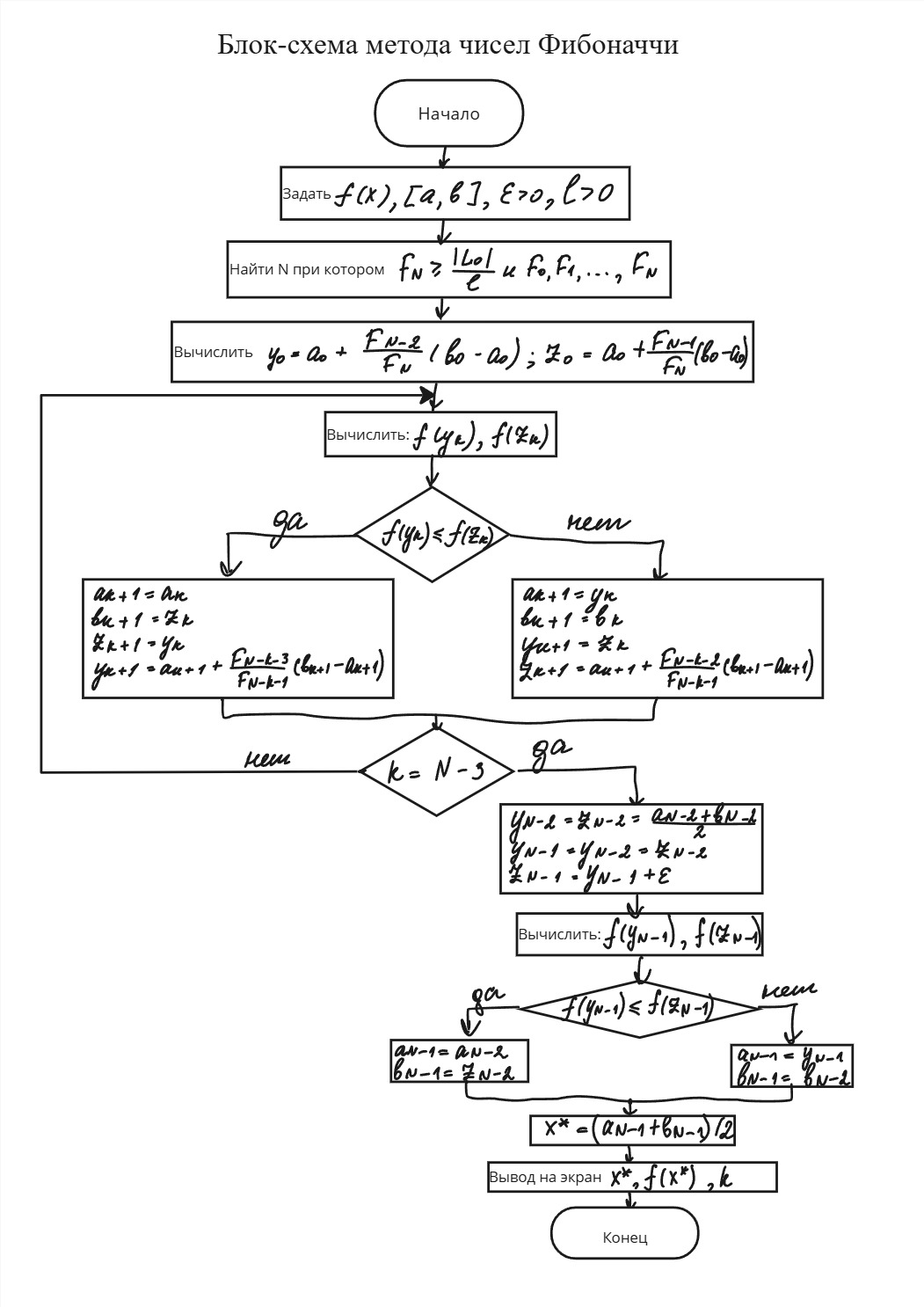




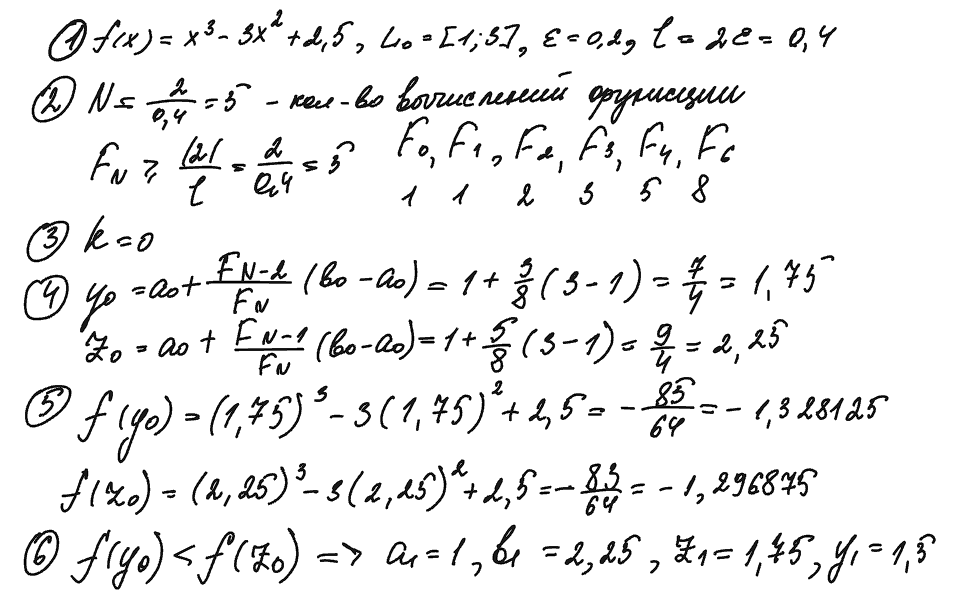


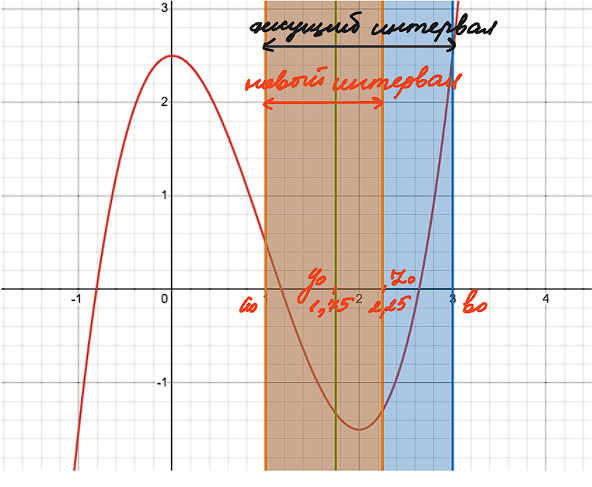


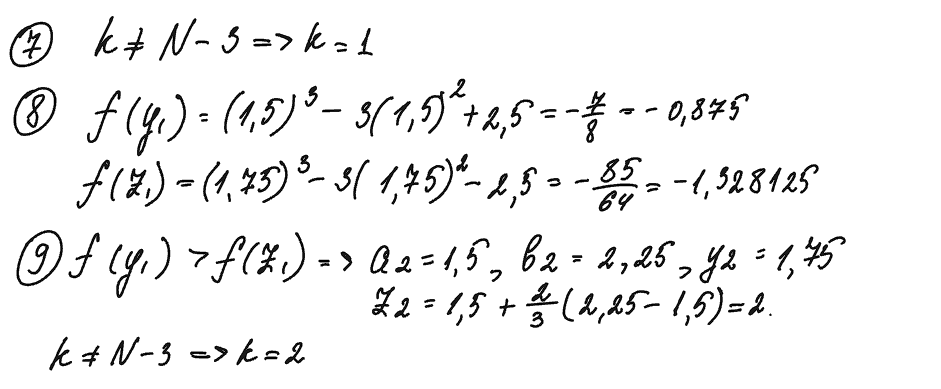
1. Построить блок-схему алгоритма поиска экстремума по методу чисел Фибоначчи.

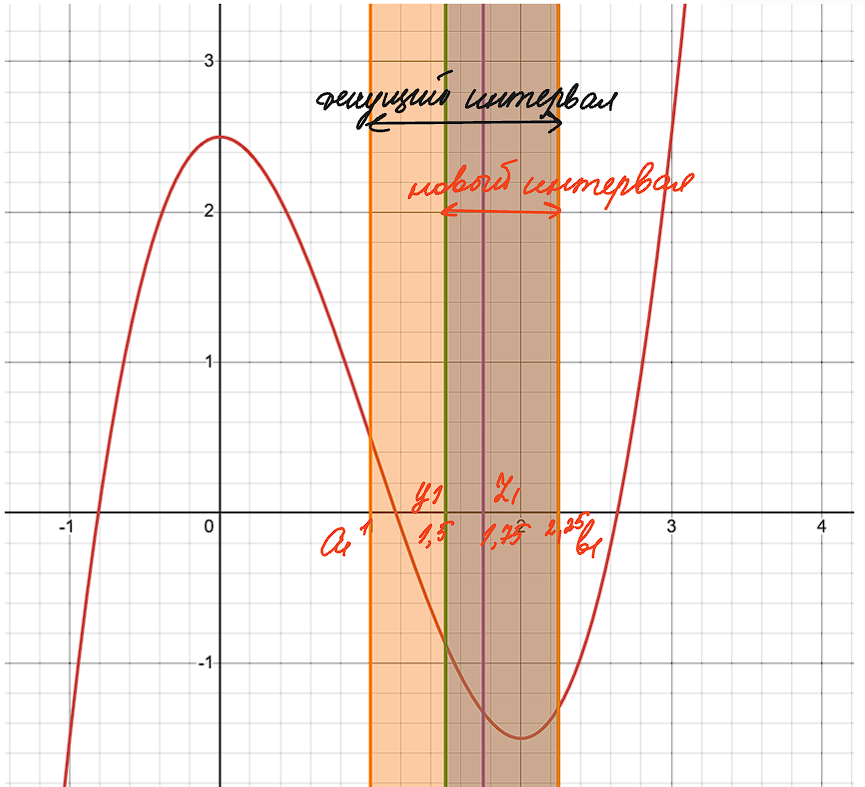


1. Найти минимум функции из индивидуального задания методом чисел Фибоначчи. Проиллюстрировать на графике функции переход от начального интервала L0 к следующему интервалу L в соответствии с методом.









1. Разработать программный модуль решения задачи индивидуального задания методом половинного деления.
2. Разработать программный модуль решения задачи индивидуального задания методом «золотого» сечения.
3. Разработать программный модуль решения задачи индивидуального задания методом чисел Фибоначчи.

Задания 6-8 реализованы в качестве одной программы с веб интерфейсом и возможностью ввода функции, выбора метода, задание интервала, задание точности, задание цели поиска (максимум или минимум). После всех введенных значений программа рассчитает координаты точки, а так же начертит график функции на заданном интервале.

Ссылка на репозиторий гит: <https://github.com/Nickolashaa/ExtremumFinder>

Код основного блока программы

**from** PyQt6**.**QtWidgets **import** QMainWindow**,** QApplication

**from** PyQt6 **import** uic

**from** PyQt6**.**QtGui **import** QPixmap

**import** sys

**from** ExtremumFinder **import** ExtremumFinder

**from** graph **import** plot\_function

class MainWindow(*QMainWindow*)**:**

**def** \_\_init\_\_(*self*)**:**

        super()**.**\_\_init\_\_()

        uic**.**loadUi('extremum.ui'**,** *self*)

*self***.**StartEngine**.**clicked**.**connect(*self***.**calculate)

*self***.**exf **=** ExtremumFinder()

**def** calculate(*self*)**:**

**try:**

            func **=** *self***.**FuncInput**.**text()

            findmin **=** 1 **if** *self***.**MinMaxInput**.**currentText() **==** "Поиск минимума" **else** 0

            a **=** float(*self***.**LeftIntervalInput**.**text())

            b **=** float(*self***.**RightIntervalInput**.**text())

**if** a **>=** b**:**

*self***.**Result**.**setText("Левая граница интервала должна быть\nменьше правой!")

**return**

            e **=** float(*self***.**EpsilonInput**.**text())

            method **=** *self***.**MethodInput**.**currentText()

*self***.**exf**.**SetFunc(func)

**if** method **==** "Метод половинчатого деления"**:**

                result**,** x**,** y **=** *self***.**exf**.**BisectionMethod(a**,** b**,** e**,** findmin)

**if** method **==** "Метод золотого сечения"**:**

                result**,** x**,** y **=** *self***.**exf**.**GoldenRatioMethod(a**,** b**,** e**,** findmin)

**if** method **==** "Метод чисел Фиббоначи"**:**

                result**,** x**,** y **=** *self***.**exf**.**FibonacciNumberMethod(a**,** b**,** e**,** findmin)

*self***.**Result**.**setText(result)

            plot\_function(*self***.**exf**.**func**,** a**,** b**,** x**,** y)

            pixmap **=** QPixmap("pic.png")

*self***.**Graph**.**setPixmap(pixmap)

**except** Exception**:**

*self***.**Result**.**setText("Заполните все поля!")

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

    app **=** QApplication(sys**.**argv)

    obj **=** MainWindow()

    obj**.**show()

    sys**.**exit(app**.**exec())

Код блока программы, который рассчитывает значение точки в зависимости от введенных ранее значений.

**from** string **import** ascii\_letters

class ExtremumFinder**:**

**def** \_\_init\_\_(*self*)**:**

*self***.**func **=** None

*self***.**variable **=** None

**def** SetFunc(*self***,** **func**)**:**

**if** '=' **in** func**:**

            func **=** func[func**.**index('=') **+** 1**:**]

        func **=** func**.**replace("^"**,** "\*\*")

        func **=** func**.**replace("e"**,** "2.71828182845904")

*self***.**func **=** func

**for** letter **in** ascii\_letters**:**

**if** letter **in** func**:**

*self***.**variable **=** letter

**break**

**def** eval(*self***,** **x**)**:**

        x\_str **=** f"({x})" **if** x **<** 0 **else** str(x)

        answer **=** *self***.**func**.**replace(*self***.**variable**,** x\_str)

**return** float(eval(answer))

**def** BisectionMethod(*self***,** **a,** **b,** **e,** **findmin**)**:**

        base\_interval **=** {

            "a"**:** a**,**

            "b"**:** b**,**

        }

        f\_a **=** *self***.**eval(a)

        f\_b **=** *self***.**eval(b)

**if** findmin**:**

**def** comparison(**a,** **b**)**:**

**return** a **<** b

**else:**

**def** comparison(**a,** **b**)**:**

**return** a **>** b

        L **=** abs(b **-** a)

**while** L **>** e**:**

            x\_sr **=** (a **+** b) **/** 2

            f\_x\_sr **=** *self***.**eval(x\_sr)

            y **=** a **+** abs(L) **/** 4

            z **=** b **-** abs(L) **/** 4

            f\_y **=** *self***.**eval(y)

            f\_z **=** *self***.**eval(z)

**if** comparison(f\_y**,** f\_x\_sr)**:**

                b **=** x\_sr

                x\_sr **=** y

**elif** comparison(f\_z**,** f\_x\_sr)**:**

                a **=** x\_sr

                x\_sr **=** z

**else:**

                a **=** y

                b **=** z

            L **=** abs(b **-** a)

        x\_ans **=** (a **+** b) **/** 2

        f\_x\_ans **=** *self***.**eval(x\_ans)

**if** comparison(f\_a**,** f\_x\_ans)**:**

            x\_ans **=** base\_interval["a"]

            f\_x\_ans **=** f\_a

**if** comparison(f\_b**,** f\_x\_ans)**:**

            x\_ans **=** base\_interval["b"]

            f\_x\_ans **=** f\_b

        result **=** f"x\* = {x\_ans:.5f}\nf(x\*) = {f\_x\_ans:.5f}"

**return** result**,** x\_ans**,** f\_x\_ans

**def** GoldenRatioMethod(*self***,** **a,** **b,** **e,** **findmin**)**:**

        base\_interval **=** {

            "a"**:** a**,**

            "b"**:** b**,**

        }

        f\_a **=** *self***.**eval(a)

        f\_b **=** *self***.**eval(b)

**if** findmin**:**

**def** comparison(**a,** **b**)**:**

**return** a **<=** b

**else:**

**def** comparison(**a,** **b**)**:**

**return** a **>** b

        y **=** a **+** ((3 **-** 5 **\*\*** 0.5) **/** 2) **\*** (b **-** a)

        z **=** a **+** b **-** y

        delta **=** None

**while** delta **is** None **or** delta **>** e**:**

            f\_y **=** *self***.**eval(y)

            f\_z **=** *self***.**eval(z)

**if** comparison(f\_y**,** f\_z)**:**

                b **=** z

                z **=** y

                y **=** a **+** b **-** y

**else:**

                a **=** y

                y **=** z

                z **=** a **+** b **-** z

            delta **=** abs(a **-** b)

        x\_ans **=** (a **+** b) **/** 2

        f\_x\_ans **=** *self***.**eval(x\_ans)

**if** comparison(f\_a**,** f\_x\_ans)**:**

            x\_ans **=** base\_interval["a"]

            f\_x\_ans **=** f\_a

**if** comparison(f\_b**,** f\_x\_ans)**:**

            x\_ans **=** base\_interval["b"]

            f\_x\_ans **=** f\_b

        result **=** f"x\* = {x\_ans:.5f}\nf(x\*) = {f\_x\_ans:.5f}"

**return** result**,** x\_ans**,** f\_x\_ans

**def** NewFibonacci(*self***,** **array**)**:**

        elem **=** array[**-**1] **+** array[**-**2]

        array**.**append(elem)

**return** elem

**def** FibonacciNumberMethod(*self***,** **a,** **b,** **e,** **findmin**)**:**

        base\_interval **=** {

            "a"**:** a**,**

            "b"**:** b**,**

        }

        f\_a **=** *self***.**eval(a)

        f\_b **=** *self***.**eval(b)

**if** findmin**:**

**def** comparison(**a,** **b**)**:**

**return** a **<=** b

**else:**

**def** comparison(**a,** **b**)**:**

**return** a **>** b

        l **=** 2 **\*** e

        L0 **=** abs(b **-** a)

        Fib **=** [1**,** 1]

**while** Fib[**-**1] **<** L0 **/** l**:**

*self***.**NewFibonacci(Fib)

        N **=** len(Fib) **-** 1

        y **=** a **+** (Fib[N**-**2] **/** Fib[N]) **\*** (b **-** a)

        z **=** a **+** (Fib[N**-**1] **/** Fib[N]) **\*** (b **-** a)

        f\_y **=** *self***.**eval(y)

        f\_z **=** *self***.**eval(z)

**for** k **in** range(N **-** 2)**:**

**if** comparison(f\_y**,** f\_z)**:**

                b **=** z

                z **=** y

                y **=** a **+** (Fib[N**-**k**-**3] **/** Fib[N**-**k**-**1]) **\*** (b **-** a)

                f\_z **=** f\_y

                f\_y **=** *self***.**eval(y)

**else:**

                a **=** y

                y **=** z

                z **=** a **+** (Fib[N**-**k**-**2] **/** Fib[N**-**k**-**1]) **\*** (b **-** a)

                f\_y **=** f\_z

                f\_z **=** *self***.**eval(z)

        z **=** y **+** e

        f\_z **=** *self***.**eval(z)

**if** comparison(f\_y**,** f\_z)**:**

            b **=** z

**else:**

            a **=** y

        x\_ans **=** (a **+** b) **/** 2

        f\_x\_ans **=** *self***.**eval(x\_ans)

**if** comparison(f\_a**,** f\_x\_ans)**:**

            x\_ans **=** base\_interval["a"]

            f\_x\_ans **=** f\_a

**if** comparison(f\_b**,** f\_x\_ans)**:**

            x\_ans **=** base\_interval["b"]

            f\_x\_ans **=** f\_b

        result **=** f"x\* = {x\_ans:.5f}\nf(x\*) = {f\_x\_ans:.5f}"

**return** result**,** x\_ans**,** f\_x\_ans

Код блока программы, который создает график на заданном интервале

**import** matplotlib**.**pyplot **as** plt

**import** numpy **as** np

**from** PIL **import** Image

**def** plot\_function(**func\_str:** *str***,** **a:** *float***,** **b:** *float***,** **point\_x:** *float***,** **point\_y:** *float***,** **n\_points=**1000)**:**

    plt**.**style**.**use('dark\_background')

    fig**,** ax **=** plt**.**subplots(**figsize=**(10**,** 6)**,** **facecolor=**'#1a1a1a')

    x **=** np**.**linspace(a**,** b**,** n\_points)

    y **=** eval(func\_str**,** {'x'**:** x**,** 'np'**:** np})

    ax**.**plot(x**,** y**,** 'b-'**,** **linewidth=**2**,** **label=**f'$f(x) = {func\_str}$')

    ax**.**axhline(0**,** **color=**'white'**,** **linewidth=**1)

    ax**.**axvline(0**,** **color=**'white'**,** **linewidth=**1)

    ax**.**grid(True**,** **linestyle=**'--'**,** **alpha=**0.5**,** **color=**'gray')

    ax**.**spines['left']**.**set\_linewidth(2)

    ax**.**spines['left']**.**set\_color('red')

    ax**.**spines['right']**.**set\_linewidth(2)

    ax**.**spines['right']**.**set\_color('red')

    ax**.**scatter(point\_x**,** point\_y**,** **color=**'lime'**,** **s=**100**,** **zorder=**5**,**

**label=**f'Point ({point\_x}, {point\_y})')

    plt**.**xlim(a**,** b)

    plt**.**savefig("pic.png"**,** **dpi=**100**,** **bbox\_inches=**'tight'**,** **transparent=**False)

    plt**.**close()

    image **=** Image**.**open("pic.png")

    image **=** image**.**resize((380**,** 280))

    image**.**save("pic.png")

Скриншоты работы программы

