

4. СИМВОЛЬНЫЕ ВЫЧИСЛЕНИЯ

[SymPy](#) – это библиотека символьной математики языка Python. Она является реальной альтернативой таким математическим пакетам как Mathematica или Maple.

Библиотека поддерживает символьные вычисления. Символьные вычисления – это преобразования и работа с математическими формулами как с последовательностью символов, которые используются в традиционных ручных методах.

С помощью обычных выражений в языке Python осуществляются численные расчеты, которые оперируют приближёнными числовыми значениями.

Например, квадратный корень числа с помощью модуля **math** в Python вычисляется вот так:

```
import math
print(math.sqrt(3))
```

Результат:

```
1.7320508075688772
```

Полученное значение является приближенным значением корня из 3, посчитанное с некоторой точностью. Если возвести это значение в квадрат, мы не получим целое число 3, так как возводим во вторую степень неточный корень из 3:

```
print(math.sqrt(3) ** 2)
```

Результат:

```
2.9999999999999996
```

Модуль SymPy позволяет получать и использовать точные значения в символьном виде. Так, пример с извлечением квадратного корня и возведением его в квадрат будет выглядеть следующим образом:

```
import sympy
print(sympy.sqrt(3))
print(sympy.sqrt(3) ** 2)
```

Результат:

```
sqrt(3)
3
```

Значение корень из 3 сохраняется в символьном виде как точное значение (`sqrt(3)`), поэтому при возведении в квадрат получается целое число 3.

Модуль SymPy преобразует выражение так, как если бы мы это делали «вручную». Например, вычислим корень из 12 «вручную»:

$$\sqrt{12} = \sqrt{4 \cdot 3} = 2 \cdot \sqrt{3}.$$

SymPy это делает аналогично, в то время как с помощью модуля **math** можно получить только приближенное число:

```
print(math.sqrt(12))
print(math.sqrt(12) ** 2)
```

Результат:

```
3.4641016151377544
11.999999999999998
```

```
print(sympy.sqrt(12))
print(sympy.sqrt(12) ** 2)
```

Результат:

```
2*sqrt(3)
12
```

Чтобы преобразовывать математические выражения и выполнять различные операции, нужно явно объявить математический символ (в понятиях языка программирования – переменную) как символьную переменную. Для этого используется оператор:

```
x = sympy.Symbol("x")
```

Теперь с помощью этого оператора можно записать различные математические выражения:

```
y = x ** 2 - x - 6
print(y)
z = (x - 1) ** 3
print(z)
```

Результат:

```
x**2 - x - 6
(x - 1)**3
```

Выражения сохраняются и обрабатываются в символьном виде. Разложим, например, выражение $x^2 - x - 6 = (x - x_1) \cdot (x - x_2)$ на множители:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} = \frac{-(-1) + \sqrt{1^2 - 4 \cdot (-6)}}{2} = \frac{1 + \sqrt{25}}{2} = 3$$

$$x_1 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} = \frac{-(-1) - \sqrt{1^2 - 4 \cdot (-6)}}{2} = \frac{1 - \sqrt{25}}{2} = -2$$

Таким образом, имеем

$$x^2 - x - 6 = (x - 3) \cdot (x + 2).$$

Для разложения на множители в SymPy используется функция **factor()**, применяя ее, получаем результат аналогичный вычисленному «в ручную»:

```
print(sympy.factor(y))
```

Результат:

```
(x - 3)*(x + 2)
```

Символьные вычисления модуля SymPy подобным образом могут использоваться для символьного интегрирования и дифференцирования, подстановки одних выражений в другие, упрощения формул и т. д.

4.1. Символьные преобразования

С помощью функций модуля **sympy** можно выполнить большое количество преобразований математических выражений. В таблице 23 приведены некоторые из них. Все они выполняются только для символьных переменных.

Таблица 23 – Функции символьных преобразований

Оператор	Назначение	Результат
<code>pprint((x-1)/(x-2))</code>	Выводит выражение в несколько строк в математическом виде	$\frac{x - 1}{x - 2}$
<code>fraction((x-1)/(x-2))</code>	Выделяет числитель и знаменатель, возвращает результат в виде списка	$(x - 1, x - 2)$
<code>expand((x-1)*(x-2))</code>	Раскрывает скобки в выражении	$x^2 - 3x + 2$
<code>factor(x**2 - x - 6)</code>	Раскладывает выражение на множители	$(x - 3)*(x + 2)$
<code>apart((2-x)/(x**2-7*x+12))</code>	Раскладывает выражение на простые дроби	$1/(x-3) - 2/(x-4)$
<code>solve(x**2 - 3*x + 2 < 0)</code>	Решает неравенство	$(1 < x) \ \& \ (x < 2)$
<code>simplify()</code>	Упрощает выражение	

Пример. Для выражения:

$$\frac{x^4 + x^3 + 2x^2 + 4x - 8}{x^5 - 3x^4 + 9x^3 - 23x^2 + 36}$$

выполнить следующие математические преобразования:

- разложить числитель и знаменатель на множители;

- разложить выражение на элементарные дроби;
- выяснить, при каких значениях x выражение больше нуля.

Код программы, необходимые пояснения приведены в комментариях:

```
# Подключить модуль sympy, указать для него имя sp:
import sympy as sp

# Объявить символьную переменную x, указать,
# что она может принимать только действительные значения.
x = sp.Symbol("x", real = True)

# Задать выражение (чтобы часть выражения можно было перенести
# на другую строку, необходимо заключить выражение в круглые скобки):
eq = (( x ** 4 + x ** 3 + 2 * x ** 2 + 4 * x - 8) /
      (x ** 5 - 3 * x ** 4 + 9 * x ** 3 - 23 * x ** 2 + 36))

# Вывести выражение в математическом виде:

print("Выражение:")
sp.pprint(eq)

# Выделить числитель и знаменатель, вывести результаты:
fract = sp.fraction(eq)
print("\nЧислитель и знаменатель:")
sp.pprint(fract)

# Занести числитель и знаменатель в переменные eq_1 и eq_2
eq_1 = fract[0]
eq_2 = fract[1]

# Разложить на множители числитель и знаменатель,
# разложить выражение на простые дроби:
f = sp.factor(eq_1)
print("\nРазложение числителя на множители: ", f)
f = sp.factor(eq_2)
print("Разложение знаменателя на множители: ", f)
f = sp.apart(eq)
print("Разложение выражения на простые дроби: ")
sp.pprint(f)

# Найти интервалы, на которых заданное выражение больше нуля.
result = sp.solve(eq > 0)
print("\nВыражение больше 0: ", result)
```

Результат:

Выражение:

$$\frac{x^4 + x^3 + 2x^2 + 4x - 8}{x^5 - 3x^4 + 9x^3 - 23x^2 + 36}$$

Числитель и знаменатель:

$$x^4 + x^3 + 2x^2 + 4x - 8, \quad x^5 - 3x^4 + 9x^3 - 23x^2 + 36$$

Разложение числителя на множители: $(x - 1)(x + 2)(x^2 + 4)$

Разложение знаменателя на множители: $(x - 2)^2(x + 1)(x^2 + 9)$

Разложение выражения на простые дроби:

$$\frac{34x + 211}{169 \cdot \left(x^2 + 9\right)} - \frac{1}{9 \cdot (x + 1)} + \frac{1384}{1521 \cdot (x - 2)} + \frac{32}{39 \cdot (x - 2)^2}$$

Выражение больше 0: $(2 < x) \mid ((-2 < x) \ \& \ (x < -1)) \mid ((1 < x) \ \& \ (x < 2))$

Задание для самопроверки

Упростить выражение:

$$f(x) = \frac{x^2 - 1}{x^3 - 1}.$$

Результат:

```
(x + 1) / (x**2 + x + 1)
```

4.2. Решение уравнений

4.2.1. Точное решение уравнений

Модуль SymPy позволяет получить точное решение уравнения одной переменной в символьном виде. Для этого используется функция `solve()`. В скобках указывается выражение, записанное для символьной переменной. Функция `solve()` вычисляет корни уравнения, правая часть которого равна 0. Например, для решения уравнения:

$$x^2 - 2 = 0$$

используется следующий оператор (переменная `x` должна быть заранее объявлена как символьная):

```
sp.solve(x ** 2 - 2)
```

В выражении можно использовать стандартные математические функции (`sin()`, `cos()`, `log()`, `exp()` и др.), определенные в модуле `sympy` (таблица 24).

Таблица 24 – Математические функции для символьных вычислений

Функция	Описание
<code>sin(x)</code>	Синус от x , используется, если x – символьная переменная
<code>cos(x)</code>	Косинус от x , используется, если x – символьная переменная
<code>tan(x)</code>	Тангенс от x , используется, если x – символьная переменная
<code>cot(x)</code>	Котангенс от x , используется, если x – символьная переменная
<code>acos(x)</code>	Арксинус от x , используется, если x – символьная переменная
<code>asin(x)</code>	Аркосинус от x , используется, если x – символьная переменная
<code>atan(x)</code>	Арктангенс от x , используется, если x – символьная переменная
<code>acot(x)</code>	Арккотангенс от x , используется, если x – символьная переменная
<code>ln(x)</code>	Натуральный логарифм x , используется, если x – символьная переменная
<code>log(x, base)</code>	Логарифм x , по основанию base , основание base может быть опущено, тогда вычисляется натуральный логарифм, используется, если x – символьная переменная
<code>exp(x)</code>	Экспонента от x , используется, если x – символьная переменная
<code>abs(x)</code>	Модуль x , используется, если x – символьная переменная
<code>sqrt(x)</code>	Корень из x , используется, если x – символьная переменная

Результат выполнения функции `solve()` – список корней. Каждый корень вычисляется в символьном виде. Например, результат решением уравнения:

$$x^2 - 2 = 0$$

является список из двух элементов:

```
[-sqrt(2), sqrt(2)]
```

Решение уравнение целесообразно сохранить в виде отдельной переменной:

```
result = sp.solve(x ** 2 - 2)
```

Тогда в переменной **result** будет храниться список решений. К отдельному корню обращение осуществляется по его индексу, например, можно вывести в консоль каждый корень по отдельности:

```
print(result[0], result[1])
```

В результате получим:

```
-sqrt(2) sqrt(2)
```

Полученные корни не являются числами, чтобы преобразовать их к числовому формату используется функция модуля `sympy N()` или метод `evalf()`. Например, преобразуем полученные корни к числовому виду при выводе:

```
print("x0 =", sp.N(result[0]), "x1 =", result[1].evalf())
```

Результат:

```
x0 = -1.41421356237310 x1 = 1.41421356237310
```

В функции `N()` и методе `evalf()` можно указать количество знаков после запятой:

```
print("x0 =", sp.N(result[0], 4), "x1 =", result[1].evalf(4))
```

Результат (в результат округляется до количества знаков после запятой на единицу меньше, чем указано в скобках):

```
x0 = -1.414 x1 = 1.414
```

Пример. Решим уравнение $f(x) = 0$, если:

$$f(x) = (x - 2)^2 - 6.$$

Код программы, необходимые пояснения приведены в комментарии:

```
# импортировать модуль символьных вычислений, дать ему имя
import sympy as sp

# опишем левую часть уравнения в виде функции:
def f(x):
    z = (x - 2) ** 2 - 6
    return z

# определить переменную, относительно которой решается уравнение,
# как символьную переменную:
x = sp.Symbol("x")

# решить уравнение в символьном виде с помощью функции solve():
result = sp.solve(f(x))

# вывести результат в виде списка:
print("Список корней:", result)

# вывести отдельные корни в символьном виде:
print("x0 =", result[0], "x1 =", result[1])

# вывести отдельные корни в числовом виде:
print("x0 =", sp.N(result[0]), "x1 =", sp.N(result[1]))
```

Результат:

```
Список корней: [2 - sqrt(6), 2 + sqrt(6)]
```

```
x0 = 2 - sqrt(6) x1 = 2 + sqrt(6)
```

```
x0 = -0.449489742783178 x1 = 4.44948974278318
```

Задания для самопроверки

1. Найти область определения функции:

$$f(x) = \frac{x^2 + 1}{x^2 - 3}.$$

Для этого необходимо:

- выделить знаменатель функции;
- приравнять знаменатель 0, решить уравнение, корни этого уравнения – точки, в которых функция не существует.

Результат:

```
Знаменатель: x**2 - 3
```

```
Точки, в которых функция не существует: [-sqrt(3), sqrt(3)]
```

2. В символьном виде решить уравнение:

$$\ln(x^2 + 1) = 1.$$

Корни вывести в символьном виде и в виде чисел.

Результат:

Список корней: $[-\sqrt{-1 + E}, \sqrt{-1 + E}]$

$x_0 = -1.31083249443209$ $x_1 = 1.31083249443209$

4.2.2. Приближенное решение уравнений

Другим способом решения уравнений является приближенное решение уравнений помощью функции **nsolve()**. Эта функция применяется в том случае, если решение точными методами в символьном виде либо невозможно, либо затруднительно. Функция для приближенного решения уравнений следующим образом:

nsolve(выражение, переменная, начальное приближение)

где **выражение** – это левая часть уравнения, которое нужно решить (правая должна быть равна нулю;

переменная – это символьная переменная, относительно которой находится решение;

начальное_приближение – начальное приближение корня (функция находит корень в окрестности этого значения).

Результат выполнения функции – число. Например, найдем приближенным методом корни уравнения:

$$x^2 - 2 = 0.$$

Начальные приближения корней обычно определяют по графику. На рисунке 33 приведен график функции $f(x) = x^2 - 2$. Как видно из графика, уравнение имеет два корня (две точки пересечения графика с осью OX), в качестве начальных приближений можно задать числа, расположенные «поблизости» от пересечений. Например, для отрицательного корня это могут быть числа: $-2, -1$.

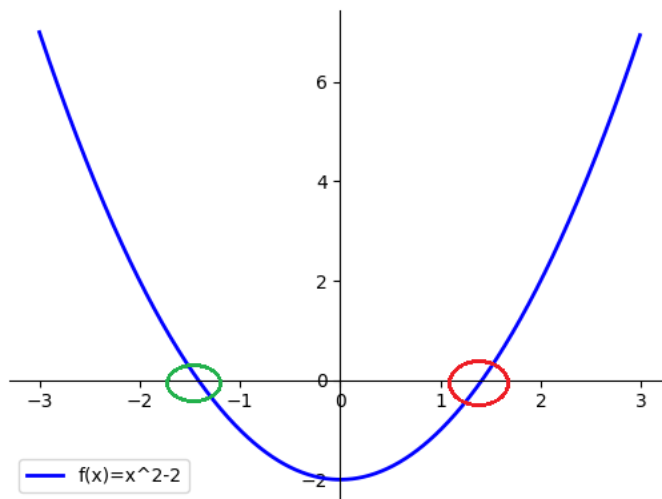


Рисунок 33. График функции

Код программы:

```
x = sp.Symbol("x")

# решаем уравнение относительно x, начальное приближение -1
x0 = sp.nsolve(x ** 2 - 2, x, -1)
print("x0 =", x0)

# решаем уравнение относительно x, начальное приближение 1
x1 = sp.nsolve(x ** 2 - 2, x, 1)
print("x1 =", x1)
```

Давайте сравним значения корня **x0**, полученные точным и приближенным методом:

```
x0_точное = -1.41421356237310  
x0_приближенное = -1.41421356237310
```

Эти значения совпадают до 14 знака после запятой. Это говорит о том, что для решения уравнений с этой точностью можно использовать оба метода. При этом приближенное решение является менее ресурсоёмким.

Пример. Решим уравнение $f(x) = 0$, если:

$$f(x) = x^3 - 6x^2 + x + 5.$$

Код программы, необходимые пояснения приведены в комментариях:

```
# сначала необходимо построить график функции, по которому определить  
# начальные приближения  
# импортируем модуль для рисования, дать ему имя  
import matplotlib.pyplot as plt  
  
# импортировать модуль символьных вычислений, дать ему имя  
import sympy as sp  
  
# опишем левую часть уравнения в виде функции  
# (если функция включает математические функции sin(), cos() и пр.  
# использовать функции из модуля sympy:  
def f(x):  
    z = x ** 3 - 6 * x ** 2 + x + 5  
    return z  
  
# интервал построения, шаг, количество точек построения 100:  
a = -2  
b = 7  
h = (b-a)/100  
# список координат по оси OX  
x_list = [a + i*h for i in range(0,101)]  
  
#список координат по оси OY, значения которого посчитаны  
y_list = [f(x) for x in x_list]  
  
#Сформируем график функции:  
plt.plot(x_list, y_list, 'b-', label="f(x)=x^3-6*x^2+x+5", lw=2)  
  
#Отформатируем оси, выведем легенду и покажем область построения:  
plt.gca().spines["left"].set_position("zero")  
plt.gca().spines["bottom"].set_position("zero")  
  
plt.gca().spines["top"].set_visible(False)  
plt.gca().spines["right"].set_visible(False)  
  
plt.legend(loc="upper center")  
  
plt.show()
```

В результате выполнения программы будет построен график, показанный на рисунке 34. По графику видно, что уравнение имеет три корня, в качестве начальных приближений можно взять числа: -2, 2, 6 соответственно.

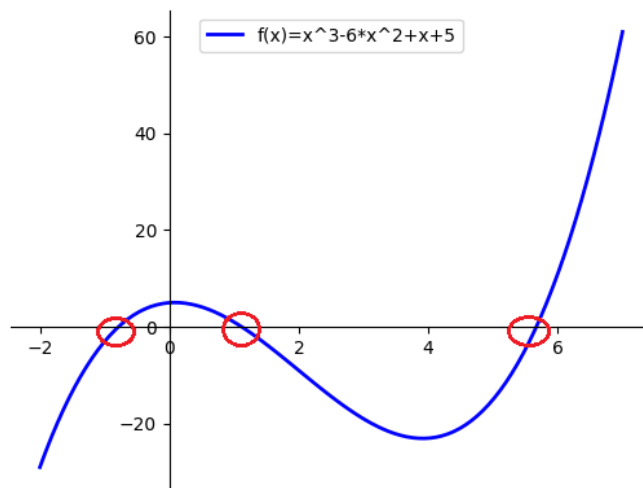


Рисунок 34. График функции

Код программы (продолжение):

```
# определить переменную, относительно которой решается уравнение,
# как символьную переменную:
x = sp.Symbol("x")

# решить уравнение с помощью функции nsolve():
x0 = sp.nsolve(f(x), x, -2)
x1 = sp.nsolve(f(x), x, 2)
x2 = sp.nsolve(f(x), x, 6)

print("x0 =", x0, "x1 =", x1, "x2 =", x2)
```

Результат:

```
x0 = -0.787759039037294 x1 = 1.11983034344540 x2 = 5.66792869559189
```

Задание для самопроверки

Приближенным методом решить уравнение, предварительно построить график:

$$\ln(x^2 + 1) = 1$$

На рисунке 35 показан график функции.

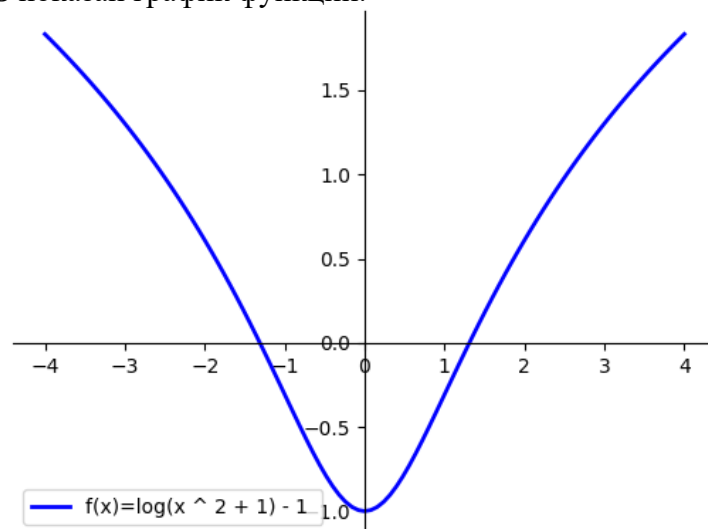


Рисунок 35. График функции

Результат:

```
x0 = -1.31083249443209 x1 = 1.31083249443209
```

4.3. Дифференцирование и интегрирование

В модуле **sympy** реализованы функции, с помощью которых можно выполнять дифференцирование и интегрирование функции. В таблице 25 приведено описание функций и примеры их использования.

Таблица 25 – Символьные операторы дифференцирования и интегрирования

Оператор	Назначение	Результат
<code>diff(4*x**2 - 3*x + 5)</code> <code>diff((4*x**2 - 3*x + 5), x)</code> <code>diff((4*x**2 - 3*x + 5), x, 1)</code>	Находит производную символьного выражения по переменной x	$8x - 3$
<code>diff((4*x**2 - 3*x + 5), x, 2)</code>	Находит вторую производную символьного выражения по переменной x	8
<code>integrate(x**2 + 2)</code> <code>integrate(x**2 + 2, x)</code>	Вычисляет неопределенный интеграл символьного выражения по переменной x	$x^3/3 + 2x$
<code>integrate(x**2 + 2, (x, 1, 2))</code>	Вычисляет определенный интеграл символьного выражения по переменной x , после которой указывается нижний и верхний предел интегрирования	13/3

Для символьных выражений можно вычислить их значения в некоторой точке. Для этого используется функция `evalf()`, в качестве аргумента указывается переменная и значение, при каком необходимо вычислить выражение, например:

```
(x ** 2 + 2).evalf(subs={x : 3})
```

Выполняется эта функция следующим образом: вместо символьной переменной (в примере это `x`) в выражение подставляется значение (в нашем случае 3) и вычисляется результат (в примере результат – 11).

4.3.1. Задача: поиск экстремумов функции

Задача. Найти локальные экстремумы функции:

$$f(x) = x^3 - 6x^2 + x + 5.$$

Пояснение. Локальный экстремум – это точки, в которых первая производная функции равна 0. Эти точки могут быть локальным минимумом (в этом случае вторая производная больше нуля), локальным максимумом (в этом случае вторая производная меньше нуля), точкой перегиба (вторая производная равна 0).

Алгоритм:

- найти первую производную функции;
- приравнять производную нулю, решить уравнение;
- найти вторую производную функции;
- вычислить значения второй производной в точках экстремума;
- определить тип точки экстремума: минимум, максимум, перегиб.

Ниже приведен код программы, необходимые пояснения даны в комментариях.

```
# Подключить модуль sympy, указать для него имя sp:
import sympy as sp

# Описать функцию
def f(x):
    z = x ** 3 - 6 * x ** 2 + x + 5
    return z
```

```

# Объявить символьную переменную x
x = sp.Symbol("x")
# Вывод функции:
print("f(x) =", f(x))
# Найти производную функции f(x) и сохранить ее в переменной d_f:
d_f = sp.diff(f(x), x)
print("первая производная функции:", d_f)

# Решить уравнение d_f = 0:
x_extr = sp.solve(d_f)
print("точки экстремума", x_extr)

# Сохранить точки экстремума в переменные:
x_0 = x_extr[0]
x_1 = x_extr[1]

# Найти вторую производную функции:
d2_f = sp.diff(f(x), x, 2)
print("вторая производная функции:", d2_f)

# Найти значение второй производной в точках экстремума x_0 и x_1,
# сохранить их в переменных f_0_d2 и f_1_d2:
f_0_d2 = d2_f.evalf(subs = {x : x_0})
f_1_d2 = d2_f.evalf(subs = {x : x_1})

# Определить, является точка x_0 точкой
# локального минимума, максимума или перегиба
if f_0_d2 > 0:
    print(x_0, "- локальный минимум")
elif f_0_d2 < 0:
    print(x_0, "- локальный максимум")
else:
    print(x_0, "- точка перегиба")

# Определить, является точка x_1 точкой
# локального минимума, максимума или перегиба
if f_1_d2 > 0:
    print(x_1, "- локальный минимум")
elif f_1_d2 < 0:
    print(x_1, "- локальный максимум")
else:
    print(x_1, "- точка перегиба")

```

Результат:

```

f(x) = x**3 - 6*x**2 + x + 5
первая производная функции: 3*x**2 - 12*x + 1
точки экстремума [2 - sqrt(33)/3, sqrt(33)/3 + 2]
вторая производная функции: 6*(x - 2)
2 - sqrt(33)/3 - локальный максимум
sqrt(33)/3 + 2 - локальный минимум

```

Задание для самопроверки

Найти максимальное и минимальное значения функции $f(x)$ на интервале $[-5, 5]$:

$$f(x) = x^3 - 6x^2 + x + 5.$$

Для этого:

- найти точки экстремума функции;
- выбрать только те из них, которые принадлежат интервалу $[-5, 5]$;
- вычислить значения функции $f(x)$ в точках экстремума и на границах интервала (в точках $-5, 5$);
- выбрать наибольшее и наименьшее значения функции.

Результат:

```
f(x) = x**3 - 6*x**2 + x + 5
точки экстремума [2 - sqrt(33)/3, sqrt(33)/3 + 2]
f( 2 - sqrt(33)/3 )= 5.04226424709296
f( sqrt(33)/3 + 2 )= -23.0422642470930
f(-5)= -275
f(5)= -15
Наименьшее значение: -275 , в точке: -5
Наибольшее значение: 5.0423 , в точке: 2 - sqrt(33)/3
```

4.3.2. Построение касательной функции

Задача. Найти касательную функции $f(x)$ в точке $x_0 = 4$ и построить ее на графике, обозначить точку касания:

$$y(x) = (x - 2)^2 - 6.$$

Пояснение. Уравнение касательной к функции $f(x)$ имеет вид:

$$y - f(x_0) = f'(x_0) \cdot (x - x_0).$$

Это прямая линия, которая описывается уравнением $y = k \cdot x + b$. В нашем случае:

$$y = f'(x_0) \cdot x + (f(x_0) - f'(x_0) \cdot x_0).$$

Следовательно, коэффициенты уравнения касательной вычисляются следующим образом:

$$k = f'(x_0), \quad b = f(x_0) - f'(x_0) \cdot x_0.$$

Алгоритм вычисления коэффициентов касательной:

- найти производную функции $f(x)$;
- вычислить значение производной в точке x_0 ;
- вычислить значение функции в точке x_0 ;
- вычислить коэффициенты k и b по формулам.

Код программы, необходимые пояснения приведены в комментариях:

```
import sympy as sp
import matplotlib.pyplot as plt

# описание функции для построения прямой
def y (x, k, b):
    z = k * x + b
    return z

# описание функции
def f (x):
    z = (x - 2) ** 2 - 6
    return z

x_0 = 3
x = sp.Symbol("x")

# вычислить производную
d_f = sp.diff(f(x), x)

# посчитать значение функции и производной в точке x_0
d_f_x_0 = d_f.evalf(subs = {x : x_0})
f_x_0 = f(x).evalf(subs = {x : x_0})

# вычислить коэффициенты касательной
k = d_f_x_0
b = f_x_0 - d_f_x_0 * x_0
print("уравнение касательной: y = %5.2f * x + %5.2f" % (k, b))
```

```

# построить график функции касательную
a = -2
a1 = 6
h = (a1 - a)/100

# список координат по оси OX
x_list = [a + i * h for i in range(0,101)]

# список координат по оси OY, значения которого посчитаны
f_list = [f(x) for x in x_list]
y_list = [y(x, k, b) for x in x_list]

# линия графика функции:
plt.plot(x_list, f_list, 'r-', label="f(x) = (x-2)^2-6", lw=2)

# линия касательной
plt.plot(x_list, y_list, 'b-', label="y = kx+b", lw=2)

# точка касания
plt.plot([x_0], [f(x_0)], 'go')

# оси, легенда
plt.gca().spines["left"].set_position("zero")
plt.gca().spines["bottom"].set_position("zero")
plt.gca().spines["top"].set_visible(False)
plt.gca().spines["right"].set_visible(False)

plt.legend()

plt.show()

```

Результат:

уравнение касательной: $y = 2.00 * x + -11.00$

Также в результате выполнения программы будет построен график функции, касательная и точка касания (рисунок 36).

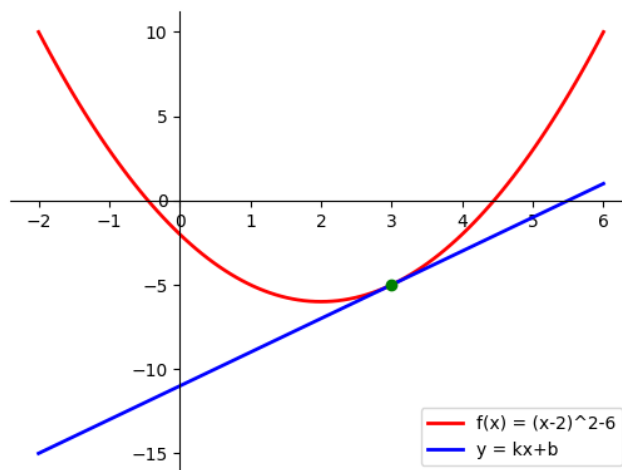


Рисунок 36. Результат выполнения программы

Задание для самопроверки

Найти уравнение касательной функции $f(x)$ в точках -2 и 3 , построить график функции, линии касательных, обозначить точки касания:

$$f(x) = x^3 - 6x^2 + x + 5.$$

Результат:

уравнение касательной в точке -0.5 : $y = 7.75 * x + 6.75$

уравнение касательной в точке 4 : $y = 1.00 * x + -27.00$

Также в результате выполнения программы должны быть построены графики функций (рисунок 37).

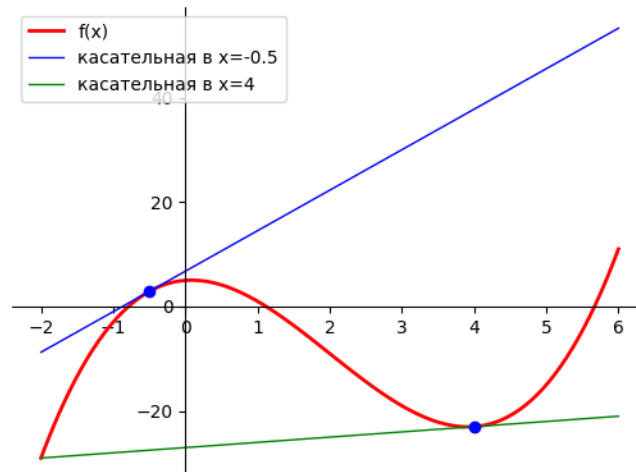


Рисунок 37. Результат выполнения программы

4.3.3. Вычисление неопределенных и определенных интегралов

Задача. Вычислить площадь области, ограниченной графиками функций $f(x)$ и $y(x)$ (рисунок 38):

$$y(x) = (x - 2)^2 - 6, \quad f(x) = x^3 - 6x^2 + x + 5.$$

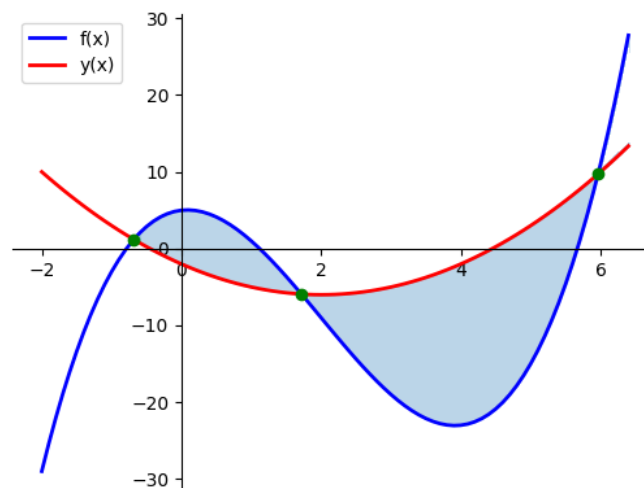


Рисунок 38. Область, ограниченная линиями графиков

Алгоритм вычисления площади:

- найти точки пересечения графиков функции (решить уравнение $f(x) = y(x)$);
- разбить область построения на две части, как показано на рисунке 38;
- найти площадь области I, здесь функция $f(x)$ больше, чем $y(x)$, область ограничена графиками функций и прямыми $x = k_0$, $x = k_1$, следовательно, площадь может быть найдена по формуле:

$$\int_{k_0}^{k_1} (f(x) - y(x)) dx;$$

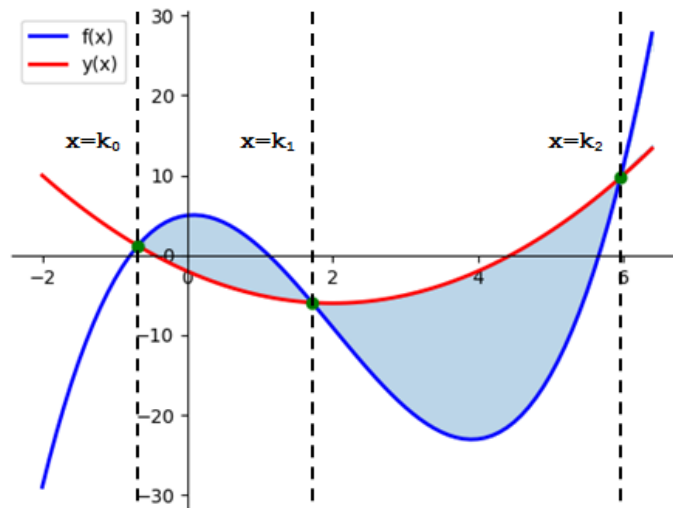


Рисунок 39. Разбиение области

- найти площадь области II, здесь функция $y(x)$ больше, чем $f(x)$, область ограничена графиками функций и прямыми $x = k_1$, $x = k_2$, следовательно, площадь может быть найдена по формуле:

$$\int_{k_1}^{k_2} (y(x) - f(x)) dx;$$

- найти общую площадь фигуры, сложив площади областей I и II.

Ниже приведен код программы, необходимые пояснения даны в комментариях:

```
import sympy as sp

# Описание функций
def y(x):
    z = (x - 2) ** 2 - 6
    return z

def f(x):
    z = x ** 3 - 6 * x ** 2 + x + 5
    return z

x = sp.Symbol("x")
# Поиск корней уравнения f(x) - y(x) = 0 приближенным методом
# для поиска начальных значений построить графики функций на одной области
k_0 = sp.nsolve(f(x) - y(x), x, -1)
k_1 = sp.nsolve(f(x) - y(x), x, 2)
k_2 = sp.nsolve(f(x) - y(x), x, 6)
print("Точки пересечения графиков k_0 = %5.3f, k_1 = %5.3f, k_2 = %5.3f" %
      (k_0, k_1, k_2))

# Вычисление площади
s_1 = sp.integrate(f(x) - y(x), (x, k_0, k_1))
s_2 = sp.integrate(y(x) - f(x), (x, k_1, k_2))
s = s_1 + s_2
print("Площадь фигуры, ограниченной графиками функций: %6.3f" % s)
```

Результат:

Точки пересечения графиков $k_0 = -0.683$, $k_1 = 1.718$, $k_2 = 5.965$
 Площадь фигуры, ограниченной графиками функций: 70.336

Задание для самопроверки

Вычислить площадь области, ограниченной графиками функций $f(x)$ и $y(x)$ (предварительно постройте графики функции):

$$y(x) = 2 \cdot x - 6, \quad f(x) = x^3 - 6x^2 + x + 5.$$

Результат:

Точки пересечения графиков $k_0 = -1.298$, $k_1 = 1.449$, $k_2 = 5.849$

Площадь фигуры, ограниченной графиками функции $f(x)$ и $y(x)$: 90.220

4.4. Предел функции

Для вычисления предела функции используется функция модуля **sympy** `limit()`.
Для вычисления предела:

$$\lim_{x \rightarrow x_0} f(x)$$

используется запись:

```
limit(f(x), x, x0)
```

В качестве x_0 можно использовать либо числовое значение, либо бесконечность. В модуле **sympy** для обозначения бесконечности служат символы «oo» (две английских прописных o):

```
limit(f(x), x, oo)
```

Для вычисления предела «справа» или «слева» используется запись:

```
limit(f(x), x, x0, '+') и limit(f(x), x, x0, '-')
```

В таблице 26 приведены примеры вычисления пределов, функция предела и константа бесконечности импортированы:

```
from sympy import limit, oo
```

Таблица 26 – Примеры вычисления пределов

Оператор	Описание	Результат
<code>limit(2*x/(4*x-2), x, oo)</code>	Вычисляет предел выражения при x , стремящемся к бесконечности	1/2
<code>limit(4/(-x**2+2*x+3), x, -1, '-')</code>	Вычисляет «левый» предел	-oo
<code>limit(4/(-x**2+2*x+3), x, -1, '+')</code>	Вычисляет «правый» предел	oo

4.4.1. Задача, поиск асимптот функции

Задача. Найти уравнения асимптот функции $f(x)$:

$$f(x) = \frac{4 \cdot x}{2 \cdot x + 3}.$$

Пояснение. Вертикальные асимптоты следует искать в точках разрыва. Прямая $x = x_b$ является вертикальной асимптотой графика функции $f(x)$, если хотя бы одно из предельных значений:

$$\lim_{x \rightarrow x_b^-} f(x) \text{ или } \lim_{x \rightarrow x_b^+} f(x)$$

равно $+\infty$ или $-\infty$.

Прямая $y = k \cdot x + b$ является наклонной асимптотой графика функции $f(x)$, если

$$\lim_{x \rightarrow \infty} |f(x) - k \cdot x - b| = 0.$$

Если наклонная асимптота существует, ее коэффициенты k и b вычисляются по формулам:

$$k = \lim_{x \rightarrow \infty} \frac{f(x)}{x}, \quad b = \lim_{x \rightarrow \infty} (f(x) - k \cdot x).$$

Горизонтальная асимптота является частным случаем наклонной, если $k = 0$.

Код программы, необходимые пояснения даны в комментариях:

```
import sympy as sp

def f(x):
    z = 4*x/(2*x+3)
    return z

x = sp.Symbol("x")

# точка разрыва
x_b = -3/2

print("Уравнение вертикальной асимптоты:")
lim = sp.limit(f(x), x, x_b)
print("предел равен: ", lim, ", уравнение асимптоты x =", x_b)

print("Уравнение наклонной асимптоты:")
k1 = sp.limit(f(x) / x, x, sp.oo)
b1 = sp.limit((f(x) - k1 * x), x, sp.oo)
print("y = %5.2f *x + %5.2f" % (k1, b1))
```

Результат:

Уравнение вертикальной асимптоты:
предел равен: $-\infty$, уравнение асимптоты $x = -1.5$
Уравнение наклонной асимптоты:
 $y = 0.00 *x + 2.00$

На рисунке 40 показан график функции и построены асимптоты.

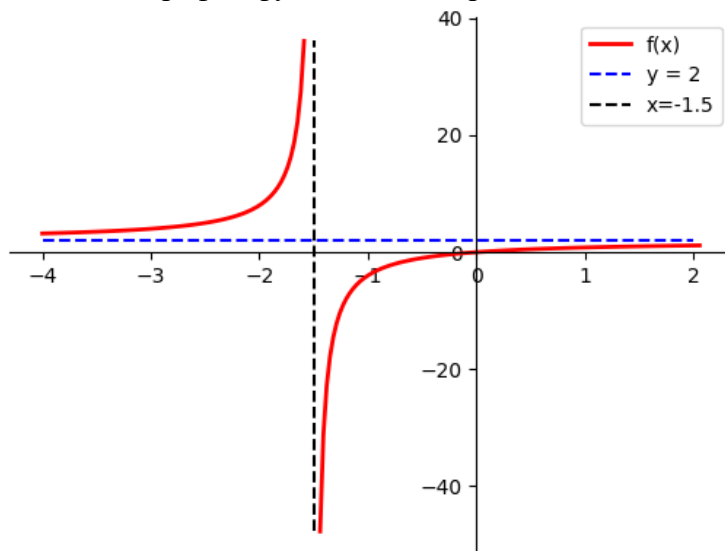


Рисунок 40. График функции и асимптоты

Задание для самопроверки

Задача. Найти уравнения асимптот функции $f(x)$:

$$f(x) = \sqrt{x^2 + 1}.$$

Результат:

Уравнения наклонных асимптот:
 $y = -1.00 *x + 0.00$
 $y = 1.00 *x + 0.00$

На рисунке 41 показан график функции и асимптоты.

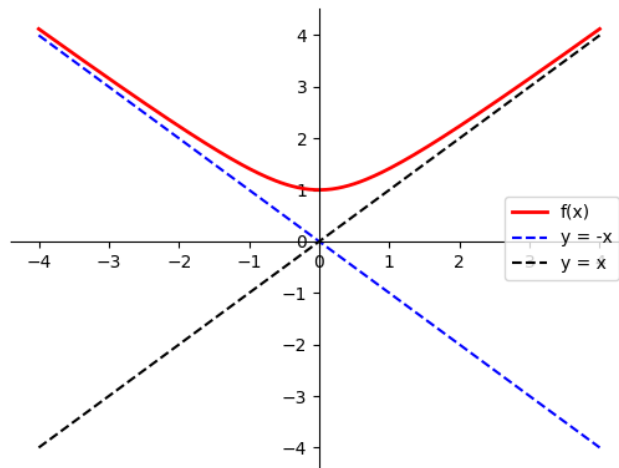


Рисунок 41. График функции и асимптоты

4.5. Задача: исследование функции и построение ее графика

Задача. Исследовать функцию $f(x)$ и построить ее график.

$$f(x) = x + \frac{1}{x-1}$$

Алгоритм:

1. Найти область определения функции, указать особые точки (точки разрыва):

- выделить знаменатель функции:

$$x - 1;$$

- приравнять знаменатель к нулю, решить уравнение:

$$x - 1 = 0.$$

Результат: функция определена на множестве действительных чисел, кроме $x = 1$, $x_b = 1$ – точка разрыва.

2. Найти точки пересечения с осями координат:

- чтобы найти точки пересечения с осью ОХ, необходимо решить уравнение:

$$x + \frac{1}{x-1} = 0;$$

- чтобы найти точку пересечения с осью ОУ, нужно вычислить значение функции в точке $x = 0$:

$$f(0).$$

Результат: точек пересечения с осью ОХ – нет, точка пересечения с осью ОУ – $(0, -1)$.

3. Найти вертикальные, горизонтальные и наклонные асимптоты:

- вертикальные асимптоты нужно искать в точках разрыва, если хотя бы одно из предельных значений:

$$\lim_{x \rightarrow x_b^-} f(x) \text{ или } \lim_{x \rightarrow x_b^+} f(x)$$

равно $+\infty$ или $-\infty$.

- если наклонная асимптота $y = k \cdot x + b$ существует, ее коэффициенты k и b вычисляются по формулам:

$$k = \lim_{x \rightarrow \infty} \frac{f(x)}{x}, b = \lim_{x \rightarrow \infty} (f(x) - k \cdot x).$$

Результат: вертикальная асимптота: $x = 1$,

наклонная асимптота: $y = 1 \cdot x + 0$.

4. Найти экстремумы функции и интервалы монотонности;

- найти первую производную функции:

$$f'(x);$$

• приравнять производную нулю, решить уравнение, корни уравнения – точки экстремума (x_{ext}):

$$f'(x) = 0;$$

• найти вторую производную функции:

$$f''(x);$$

• вычислить значения второй производной в точках экстремума:

$$f''(x_{ext});$$

• определить тип точки экстремума:

- если $f''(x_{ext}) > 0$, то x_{ext} – минимум,

- если $f''(x_{ext}) < 0$, то x_{ext} – максимум,

- если $f''(x_{ext}) = 0$, то x_{ext} – перегиб;

• найти интервалы возрастания и убывания функции:

- если на промежутке производная функции положительна, то функция возрастает на этом промежутке;

- если на промежутке производная функции отрицательна, то функция убывает.

Результат: точки экстремума $x_{0extr} = 0$, $x_{1extr} = 2$

$x_{0extr} = 0$ – локальный максимум,

$x_{1extr} = 2$ – локальный минимум,

функция возрастает на интервале $(-\infty; 0) \cup (2, +\infty)$,

функция убывает на интервале $(0; 1) \cup (1; 2)$.

5. Исследовать точки перегиба, найти и интервалы выпуклости/вогнутости:

• исследовать знак второй производной при переходе через точку перегиба:

- если слева (или справа) от точки перегиба значение второй производной больше 0, то на этом интервале функция вогнута;

- если слева (или справа) от точки перегиба значение второй производной меньше 0, то на этом интервале функция выпукла.

Результат: точек перегиба нет.

6. Построить график:

• описать функцию, включить в описание обработку исключений;

• определить количество точек построения n ;

• выбрать интервал построения графика $[x_{beg}, x_{end}]$ так, чтобы в него вошли все вычисленные ранее точки (особые точки, экстремумы, точки пересечения с осями) с «запасом»;

• если функция имеет точки разрыва, то разбить интервал на несколько (если одна точка разрыва – то на два, две – на три и т.д.):

$$[x_{beg}, x_b - \varepsilon] \cup [x_b + \varepsilon, x_{end}],$$

ε – подобрать произвольно;

• вычислить шаг h по формуле:

$$h = \frac{x_{end} - x_{beg}}{n - 1};$$

• вычислить количество точек построения, расположенных до точки разрыва:

$$n_1 = \text{int} \left(\frac{x_b - x_{beg}}{h} \right) - 1;$$

• вычислить количество точек построения, расположенных после точки разрыва:

$$n_2 = n - n_1 - 2;$$

• сформировать список (списки) значений изменения абсцисс графика;

• сформировать списки (списки) значений изменения ординат графика;

• построить линию графика для каждой пары списков (абсцисс и ординат).

7. Построить асимптоты:

• для построения вертикальной асимптоты необходимо найти максимальное и минимальное значения ординат, вычисленные по списку (спискам) ординат, тогда для

построения вертикальной асимптоты можно использовать две точки, занесенные в списки абсцисс ($[x_b, x_b]$) и ординат ($[min_f, max_f]$);

- для построения наклонной асимптоты необходимо определить линейную функцию, параметрами которой являются значение x , а также коэффициенты прямой k и b .

8. Обозначить на графике экстремумы функции и точки пересечения с осями.

Код программы, необходимые пояснения приведены в комментариях:

```
import sympy as sp
import matplotlib.pyplot as plt

# линейная функция
def y (x, k, b):
    z = k * x + b
    return z

# функция для исследования
def f(x):
    try:
        z = x + 1/ (x - 1)
    except:
        y = sp.oo
    return z

# символьная переменная, определена для действительных чисел
x = sp.Symbol("x", real=True)

print("Функция:", f(x))

# Область определения:
print("\nОбласть определения:")
fract = sp.fraction(sp.factor(f(x)))
denom_f = fract[1]
print(" знаменатель функции: ", denom_f)
x_solve = sp.solve(denom_f)
x_b = x_solve[0]
print(" функция не определена в точке: ", x_b)

# Точки пересечения с осями
print("\nТочки пересечения с осями:")
result = sp.solve(f(x))
if result == []:
    print(" - точек пересечения с осью OX нет")
else:
    print(" - точки пересечения с осью OX: x =", result)
print(" - точка пересечения с осью OY: y =", f(0))

# Уравнения асимптот
print("\nУравнения асимптот:")
lim = sp.limit(f(x), x, x_b, '+')
if abs(lim) == sp.oo:
    print(" - вертикальная асимптота: x =", x_b)
else:
    print(" - x =", x_b, "не является вертикальной асимптотой")

k = sp.limit(f(x) / x, x, sp.oo)
b = sp.limit((f(x) - k * x), x, sp.oo)
print(" - наклонная асимптота: y = %5.2f *x + %5.2f" % (k, b))

# Точки экстремума и интервалы монотонности
print("\nТочки экстремума и интервалы монотонности:")
# - производная функции f(x):
d_f = sp.diff(f(x), x)
```

```

# - решение уравнения d_f = 0:
x_extr = sp.solve(d_f)
print(" - точки экстремума и интервалы монотонности", x_extr)

# - точки экстремума:
x_0 = x_extr[0]
x_1 = x_extr[1]

# - вторая производная функции:
d2_f = sp.diff(f(x), x, 2)

# - значение второй производной в точках экстремума x_0 и x_1:
f_0_d2 = d2_f.evalf(subs = {x : x_0})
f_1_d2 = d2_f.evalf(subs = {x : x_1})

# - исследование точки x_0
# - локальный минимум, максимум или перегиб
if f_0_d2 > 0:
    print("\t", x_0, "- локальный минимум")
elif f_0_d2 < 0:
    print("\t", x_0, "- локальный максимум")
else:
    print("\t", x_0, "- точка перегиба")

# - исследование точки x_1
# - локальный минимум, максимум или перегиб
if f_1_d2 > 0:
    print("\t", x_1, "- локальный минимум")
elif f_1_d2 < 0:
    print("\t", x_1, "- локальный максимум")
else:
    print("\t", x_1, "- точка перегиба")

# - интервалы монотонности
print(" - интервалы монотонности:")
inc_f = sp.solve(d_f > 0)
dec_f = sp.solve(d_f < 0)
print("\tфункция возрастает на:", inc_f)
print("\tфункция убывает на:", dec_f)
# Построение графика функции
n = 200
x_beg = -2
x_end = 4
h = (x_end - x_beg) / (n - 1)
n_1 = int((x_b - x_beg) / h) - 1
n_2 = n - n_1 - 2

# список со значениями аргумента x до точки разрыва:
x_list_1 = [x_beg + h * i for i in range(n_1)]

# список со значениями аргумента x после точки разрыва:
x_list_2 = [x_b + 2*h + h * i for i in range(n_2)]

# списки со значениями двух частей функций f(x):
f_list_1 = [f(x) for x in x_list_1]
f_list_2 = [f(x) for x in x_list_2]

# линии графиков функций:
line_f_1 = plt.plot(x_list_1, f_list_1, label='f(x)')
line_f_2 = plt.plot(x_list_2, f_list_2)

# стили линий:
plt.setp(line_f_1, color="red", linewidth=2)
plt.setp(line_f_2, color="red", linewidth=2)

```

```

# Построение асимптот
# - наклонная асимптота
# список со значениями аргумента x на всем интервале:
x_list = [x_beg + h * i for i in range(n)]

# список значений линейной функции
f_list_a_o = [y(x, k, b) for x in x_list]

# линия наклонной асимптоты
plt.plot(x_list, f_list_a_o, "k--", label='асимптота y = ' + str(k) + 'x + ' + str(b))

# - вертикальная асимптота
# максимальное и минимальное значение функции по двум спискам аргументов
min_f = min(min(f_list_1), min(f_list_2))
max_f = max(max(f_list_1), max(f_list_2))
# линия вертикальной асимптоты
plt.plot([x_b, x_b], [min_f, max_f], "b--", label='асимптота x=' + str(x_b))

# - точки экстремумов
plt.plot([x_0, x_1], [f(x_0), f(x_1)], "go", label='экстремумы')

#Отформатируем оси, выведем легенду и покажем область построения:
plt.gca().spines["left"].set_position("zero")
plt.gca().spines["bottom"].set_position("zero")

plt.gca().spines["top"].set_visible(False)
plt.gca().spines["right"].set_visible(False)

plt.legend()
# loc="upper center"
plt.show()

```

Результат:

Функция: $x + 1/(x - 1)$

Область определения:

знаменатель функции: $x - 1$

функция не определена в точке: 1

Точки пересечения с осями:

- точек пересечения с осью OX нет
- точка пересечения с осью OY: $y = -1.0$

Уравнения асимптот:

- вертикальная асимптота: $x = 1$
- наклонная асимптота: $y = 1.00 * x + 0.00$

Точки экстремума и интервалы монотонности:

- точки экстремума и интервалы монотонности [0, 2]
 - 0 - локальный максимум
 - 2 - локальный минимум
- интервалы монотонности:
 - функция возрастает на: $(2 < x) \mid (x < 0)$
 - функция убывает на: $(x > 0) \& (x < 2) \& \mathbb{N}(x, 1)$

Также в результате выполнения программы будет построен график, показанный на рисунке 42.

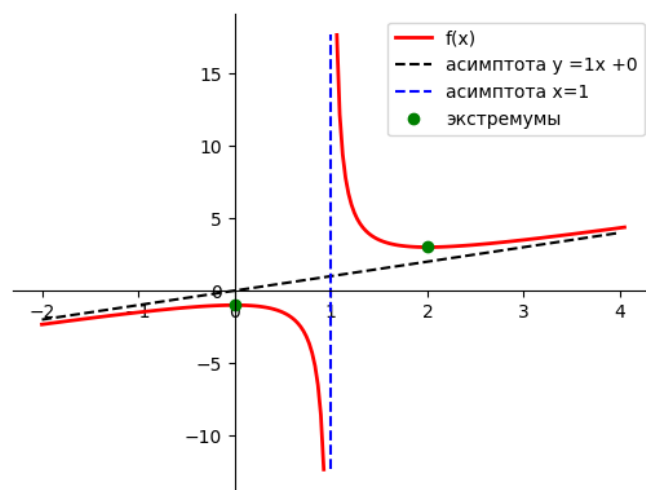


Рисунок 42. Результат выполнения программы