МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Московский авиационный институт
(национальный исследовательский университет)

Институт № 8
Компьютерные науки и прикладная математика

Кафедра 806 «Вычислительная математика и программирование»

Курсовой проект

по дисциплине «Численные методы»

Вариант №11

Выполнил: студент группы М8О-311Б-20

Комиссаров Антон Сергеевич

(Фамилия, имя, отчество)

Преподаватель:

Киндинова Виктория Валерьевна

(Фамилия, имя, отчество)

(подпись)

Оценка: **Дата**:

Москва, 2022

Содержание

1 Задание №1

Условие

Методом Гаусса вычислить решение СЛАУ $A * \overline{x} = \overline{b}$:

а) $$A = \begin{pmatrix} 11 & 5 & 2 \\ 5 & 11 & -11 \\ 2 & -11 & 11 \end{pmatrix} \qquad \overline{b} = \begin{pmatrix} 1 \\ 11 \\ -11 \end{pmatrix}$$

б) $$A = \begin{pmatrix} 16 & 0 & 11 & 11 \\ 3 & 3 & 0 & 11 \\ 2 & 5 & 3 & 11 \\ 11 & 22 & 11 & 1 \end{pmatrix} \qquad \overline{b} = \begin{pmatrix} 2 \\ 2 \\ 2 \\ 11 \end{pmatrix}$$

Алгоритм

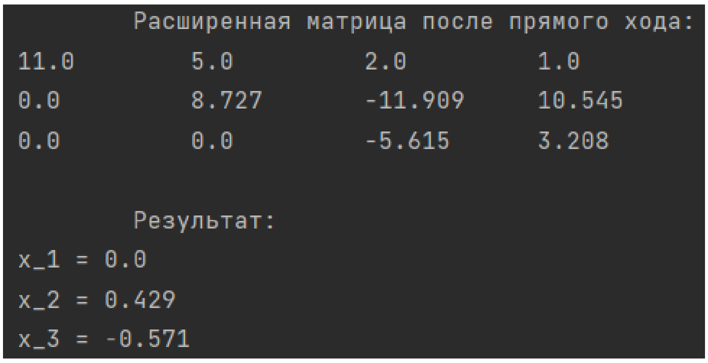
Прямым ходом приводим расширенную матрицу *big_matrix* к верхнетреугольному виду, обнуляя элементы, которые лежат ниже главной диагонали.

```
for main_index in range(matrix_size - 1):
   for i in range(main_index + 1, matrix_size):
      coefficient = big_matrix[i][main_index] / big_matrix[main_index][main_index]
      for j in range(main_index, matrix_size + 1):
         big_matrix[i][j] -= big_matrix[main_index][j] * coefficient
```

Обратным ходом вычисляем неизвестные и записываем их в список *result*.

```
result = [0] * matrix_size
   for i in range(matrix_size - 1, -1, -1):
      result[i] = big_matrix[i][matrix_size]
      for j in range(matrix_size - 1, i, -1):
         result[i] -= big_matrix[i][j] * result[j]
      result[i] /= big_matrix[i][i]
```

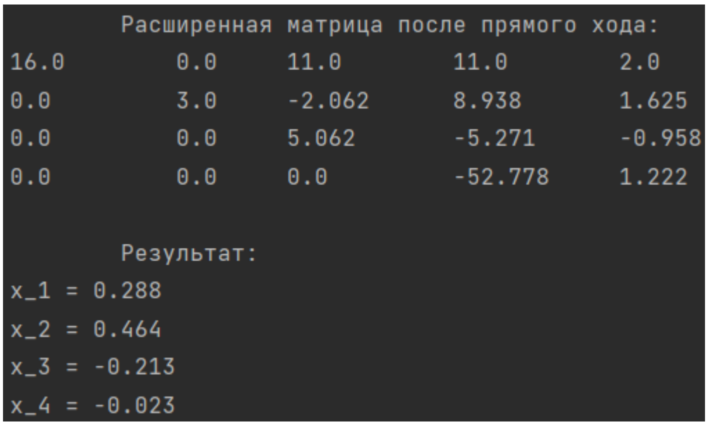
Результат



```
        Расширенная матрица после прямого хода:
11.0        5.0         2.0         1.0
0.0         8.727       -11.909     10.545
0.0         0.0         -5.615      3.208

        Результат:
x_1 = 0.0
x_2 = 0.429
x_3 = -0.571
```

Рис. 1: Пункт *а*



```
        Расширенная матрица после прямого хода:
16.0        0.0     11.0        11.0        2.0
0.0         3.0     -2.062      8.938       1.625
0.0         0.0     5.062       -5.271      -0.958
0.0         0.0     0.0         -52.778     1.222

        Результат:
x_1 = 0.288
x_2 = 0.464
x_3 = -0.213
x_4 = -0.023
```

Рис. 2: Пункт *б*

2 Задание №2

Условие

Методом прогонки вычислить решение СЛАУ с расширенной матрицей порядка а) $n = 5$; б) $n = 10$ ($N_г = N_с = 11$):

$$
\left(\begin{array}{cccccccccc|c}
b_1 & c_1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & d_1 \\
a_2 & b_2 & c_2 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & d_2 \\
0 & a_3 & b_3 & c_3 & 0 & \dots & 0 & 0 & 0 & 0 & d_3 \\
0 & 0 & a_4 & b_4 & c_4 & \dots & 0 & 0 & 0 & 0 & d_4 \\
0 & 0 & 0 & a_5 & b_5 & \dots & 0 & 0 & 0 & 0 & d_5 \\
\dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\
0 & 0 & 0 & 0 & 0 & \dots & a_{n-2} & b_{n-2} & c_{n-2} & 0 & d_{n-2} \\
0 & 0 & 0 & 0 & 0 & \dots & 0 & a_{n-1} & b_{n-1} & c_{n-1} & d_{n-1} \\
0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & a_n & b_n & d_{n-1}
\end{array}\right)
$$

$$a_i = i \cdot Nc + Nг, \quad b_i = Nc \cdot i \cdot i + Nг, \quad c_i = Nг - Nc \cdot i \quad , \quad d_i = Nc + Nг \cdot i.$$

Алгоритм

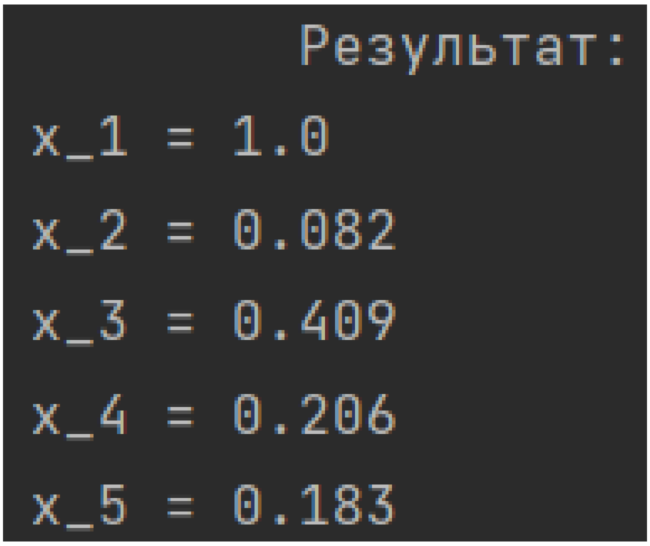
Прямым ходом вычисляем прогоночные коэффициенты P_i и Q_i.

```
P, Q = [-C[0] / B[0]], [D[0] / B[0]]
for i in range(1, count):
    denominator = A[i] * P[i-1] + B[i]
    P.append(-C[i] / denominator)
    Q.append((D[i] - A[i] * Q[i-1]) / denominator)
```

Обратным ходом вычисляем все неизвестные и записываем их в список *result*.

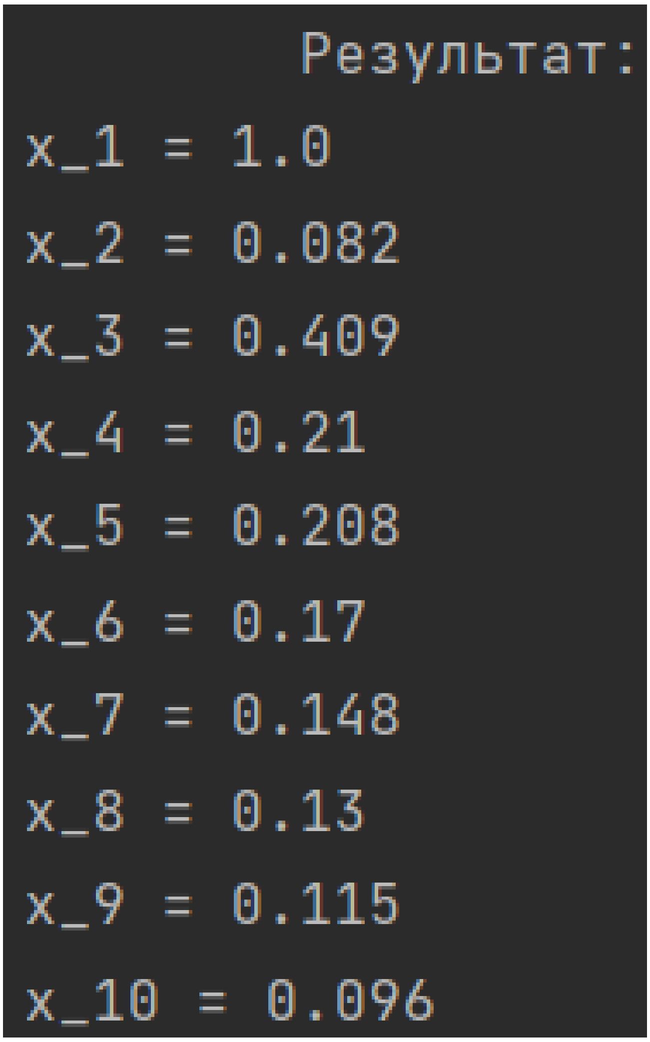
```
result = [0] * (count - 1) + [Q[-1]]
for i in range(count - 2, -1, -1):
    result[i] = P[i] * result[i + 1] + Q[i]
```

Результат



```
Результат:
x_1 = 1.0
x_2 = 0.082
x_3 = 0.409
x_4 = 0.206
x_5 = 0.183
```

Рис. 3: Пункт *а*



```
Результат:
x_1 = 1.0
x_2 = 0.082
x_3 = 0.409
x_4 = 0.21
x_5 = 0.208
x_6 = 0.17
x_7 = 0.148
x_8 = 0.13
x_9 = 0.115
x_10 = 0.096
```

Рис. 4: Пункт *б*

3 Задание №3

Условие

Методом простых итерацийрешить СЛАУ $B * \overline{x} = \overline{c}$:

а) $$B = \begin{pmatrix} 21 & 11 & 1 \\ 11 & 21 & 3 \\ 1 & 3 & 15 \end{pmatrix} \qquad \overline{c} = \begin{pmatrix} 11 \\ 21 \\ 0 \end{pmatrix}$$

б) $$B = \begin{pmatrix} 21 & 11 & 1 & 1 \\ 11 & 21 & 2 & 2 \\ 1 & 2 & 15 & 1 \\ 1 & 1 & 1 & 22 \end{pmatrix} \qquad \overline{c} = \begin{pmatrix} 11 \\ 21 \\ 0 \\ 1 \end{pmatrix}$$

Алгоритм

Разделим каждую строку на соответствующий коэффициент, находящийся на главной диагонали и взятый с обратным знаком. После этого обнуляем выбранный коэффициент.

```
for i in range(count):
    element = matrix[i][i]
    for j in range(count):
        matrix[i][j] /= -element
    vector[i] /= element
    matrix[i][i] = 0
```

Вычислим нормы матрицы B и вектора $\overline{c}$. Норма матрицы и норма вектора должны быть согласованы.

```
norm_values = [[1, max(sum(abs(element) for element in string) for string in matrix)], [2, 0], [3,
↪  math.sqrt(sum(sum(element * element for element in string) for string in matrix))]]
for i in range(count):
    column_sum = 0
    for j in range(count):
        column_sum += abs(matrix[j][i])
    norm_values[1][1] = max(norm_values[1][1], column_sum)
norm_values = min(norm_values, key=lambda el: el[0])
matrix_norm = norm_values[1]

```

```python
coefficient = 0
if matrix_norm < 1:
    if norm_values[0] == 1:
        coefficient = max(abs(element) for element in vector)
    elif norm_values[0] == 2:
        coefficient = sum(abs(element) for element in vector)
    else:
        coefficient = math.sqrt(sum(element * element for element in vector))
    coefficient /= (1 - matrix_norm)
```

Вычисляем значения неизвестных до тех пор, пока не достигнем заданной точности.

```python
x = vector.copy()
while True:
    new_x = vector.copy()
    for i in range(count):
        for j in range(count):
            if i != j:
                new_x[i] += matrix[i][j] * x[j]
    if matrix_norm < 1:
        coefficient *= matrix_norm
    else:
        sub_x = [new_x[i] - x[i] for i in range(len(x))]
        coefficient = min(max(abs(element) for element in sub_x), sum(abs(element) for element in
        ↪  sub_x), math.sqrt(sum(element * element for element in sub_x)))
    if coefficient < precision:
        break
    x = new_x.copy()
```

Результат

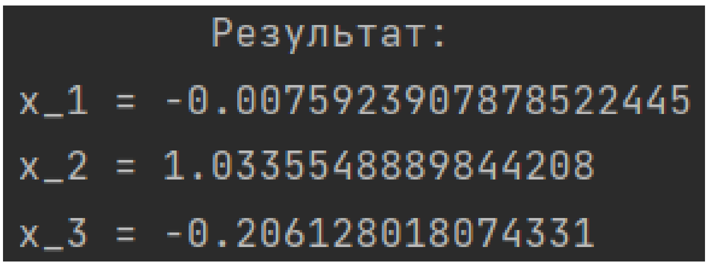


Рис. 5: Пункт *а*

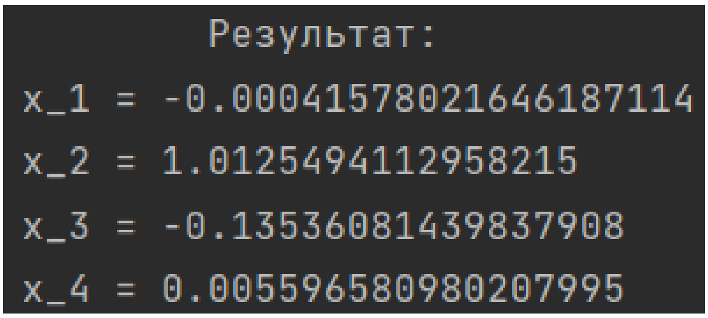


Рис. 6: Пункт *б*

4 Задание №4

Условие

Методом вращений Якоби найти все собственные числа и собственные векторы матрицы B.

а) $$B = \begin{pmatrix} 21 & 11 & 1 \\ 11 & 21 & 3 \\ 1 & 3 & 15 \end{pmatrix}$$

б) $$B = \begin{pmatrix} 21 & 11 & 1 & 1 \\ 11 & 21 & 2 & 2 \\ 1 & 2 & 15 & 1 \\ 1 & 1 & 1 & 22 \end{pmatrix}$$

Алгоритм

Выбираем максимальный по модулю недиагональный элемент. Затем находим ортогональную матрицу $U^{(k)}$, где на месте максимального по модулю элемента стоит 0. Выполняем умножение транспонированной матрицы поворота *reversed_matrix*, основной матрицы *matrix* и матрицы поворота *rotation_matrix*. Применяем условие окончания итерационного процесса.

```
result_matrix = None
while True:
    max_i, max_j = max([max([(abs(matrix[i][j]), i, j) for j in range(i + 1, count)], key=lambda el:
     ↪  el[0]) for i in range(count - 1)], key=lambda el: el[0])[1:]
    rotation_matrix = [[0.0] * count for i in range(count)]
    for i in range(count):
        rotation_matrix[i][i] = 1.0
    if matrix[max_i][max_i] == matrix[max_j][max_j]:
        angle = math.pi / 4 if matrix[max_i][max_j] > 0 else -math.pi / 4
    else:
        angle = 0.5 * math.atan(2 * matrix[max_i][max_j] / (matrix[max_i][max_i] -
         ↪  matrix[max_j][max_j]))
    rotation_matrix[max_i][max_j] = -math.sin(angle)
    rotation_matrix[max_j][max_i] = math.sin(angle)
    rotation_matrix[max_i][max_i] = rotation_matrix[max_j][max_j] = math.cos(angle)

```

```python
    reversed_matrix = [element.copy() for element in rotation_matrix]
    for i in range(count - 1):
        for j in range(i + 1, count):
            reversed_matrix[i][j], reversed_matrix[j][i] = reversed_matrix[j][i], reversed_matrix[i][j]

    result = multiplication(multiplication(reversed_matrix, matrix, count), rotation_matrix, count)
    matrix = [element.copy() for element in result]

    if result_matrix is None:
        result_matrix = [element.copy() for element in    rotation_matrix]
    else:
        result_matrix = multiplication(result_matrix, rotation_matrix, count)

    calc_E = math.sqrt(sum(sum(matrix[i][j] * matrix[i][j] for j in range(i + 1, count)) for i in
    ↪  range(count - 1)))
    if calc_E < precision:
        break
```

Результат

```
        Результат:
λ_1 = 32.460524956560626, вектор: [1.0, 1.0207163068658052, 0.23264753641963226]
λ_2 = 9.603147766929444, вектор: [-0.9315495113296278, 1.0, -0.38326988932883077]
λ_3 = 14.936327276509937, вектор: [-0.3197878223948923, 0.08537169964770298, 1.0]
```

Рис. 7: Пункт *а*

```
        Результат:
λ_1 = 32.58361858474987, вектор: [1.0, 1.0083595874002933, 0.1887889907625009, 0.302874495496O231]
λ_2 = 9.904054784710452, вектор: [-0.9530059110876193, 1.0, -0.1859383569227757, -0.06686123086072511]
λ_3 = 14.745999441342075, вектор: [-0.16977250932118695, 0.01631452779277966, 1.0, -0.11710260349583526]
λ_4 = 21.7663271891976, вектор: [-0.20497810445090714, -0.11283854252852457, 0.08414386388548271, 1.0]
```

Рис. 8: Пункт *б*

5 Задание №5

Условие

С заданной точностью $\varepsilon = 10^{-3}$ методом дихотомии найти отрицательный корень уравнения; методом итераций и методом Ньютона (касательных) найти корень уравнения:

$$\left(\frac{x}{11}\right)^3 - \frac{x}{11} + \frac{1}{11} = 0$$

Алгоритм

Реализуем метод дихотомии. Делим исходный отрезок пополам до тех пор, пока его длина не станет меньше удвоенной погрешности ε. Корень уравнения – середина конечного отрезка.

```
if function(a) * function(b) > 0:
   print(f'Некорректные a и b: {a}, {b}')
   return
in_a, in_b = a, b
while not abs(a - b) < 2 * E:
   fa = function(a)
   mid_x = (a + b) / 2
   mid_y = function(mid_x)
   if not mid_y:
      break
   elif fa * mid_y < 0:
      b = mid_x
   else:
      a = mid_x
```

Реализуем метод итераций. С помощью эквивалентного уравнения находим каждый последующий элемент последовательности до тех пор, пока не будет достигнута заданная погрешность ε. Корень уравнения – последний элемент последовательности.

```
q = 0.33
e_coefficient = q/(1 - q)
x = 0
new_x = (a + b) / 2
while not abs(new_x - x)*e_coefficient < E:
   x = new_x
   new_x = equal_func(x)
```

Реализуем метод Ньютона (касательных). Строим касательные к графику до тех пор, пока отношение значения функции к производной не станет меньше заданной погрешности ε. Корень уравнения – последняя вычисленная абсцисса.

```
if not function(a) * function(b) < 0 or not function(b) * second_derivative(b) > 0:
    print("Не выполняется условие сходимости!")
    return
x = 0
new_x = b
while not abs(new_x - x) < E:
    x = new_x
    y = function(x)
    new_x = x - (y / first_derivative(x))
```

Графики

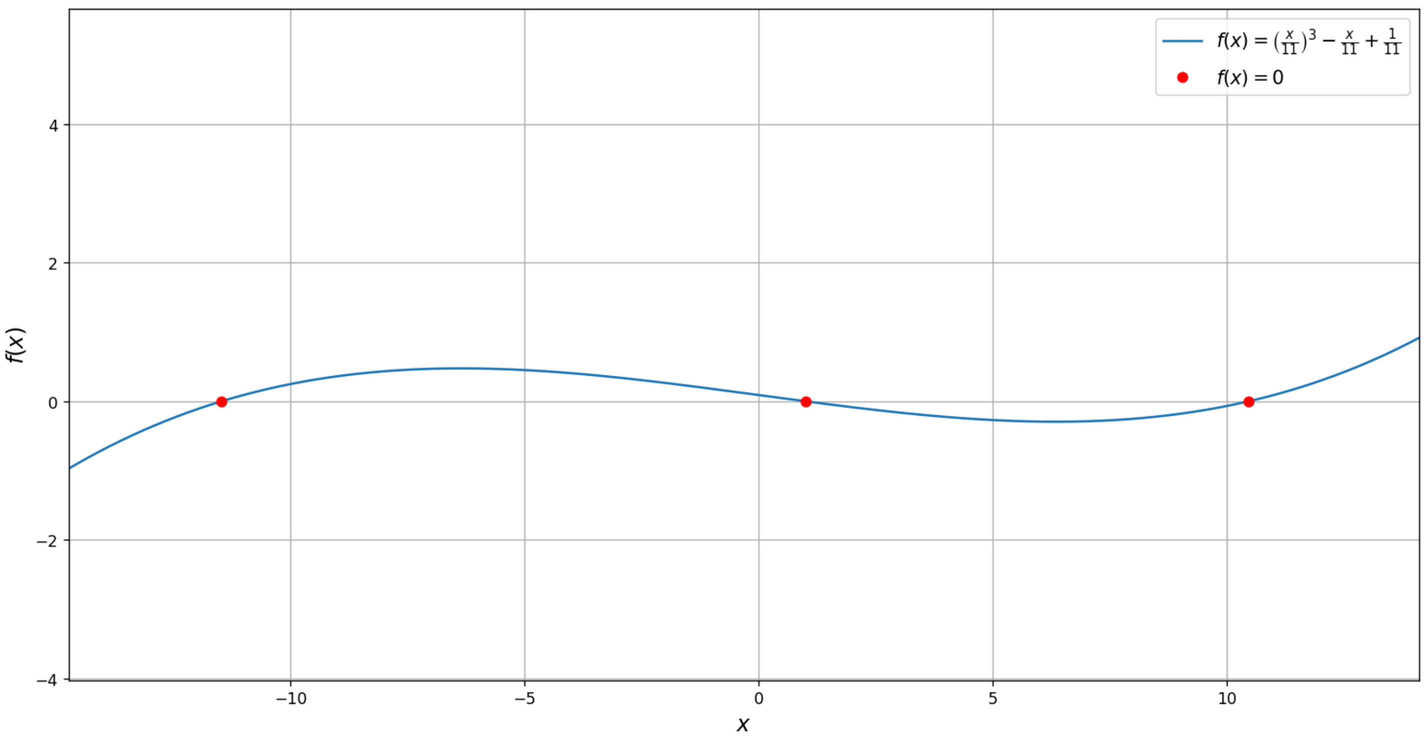


Рис. 9: Метод дихотомии

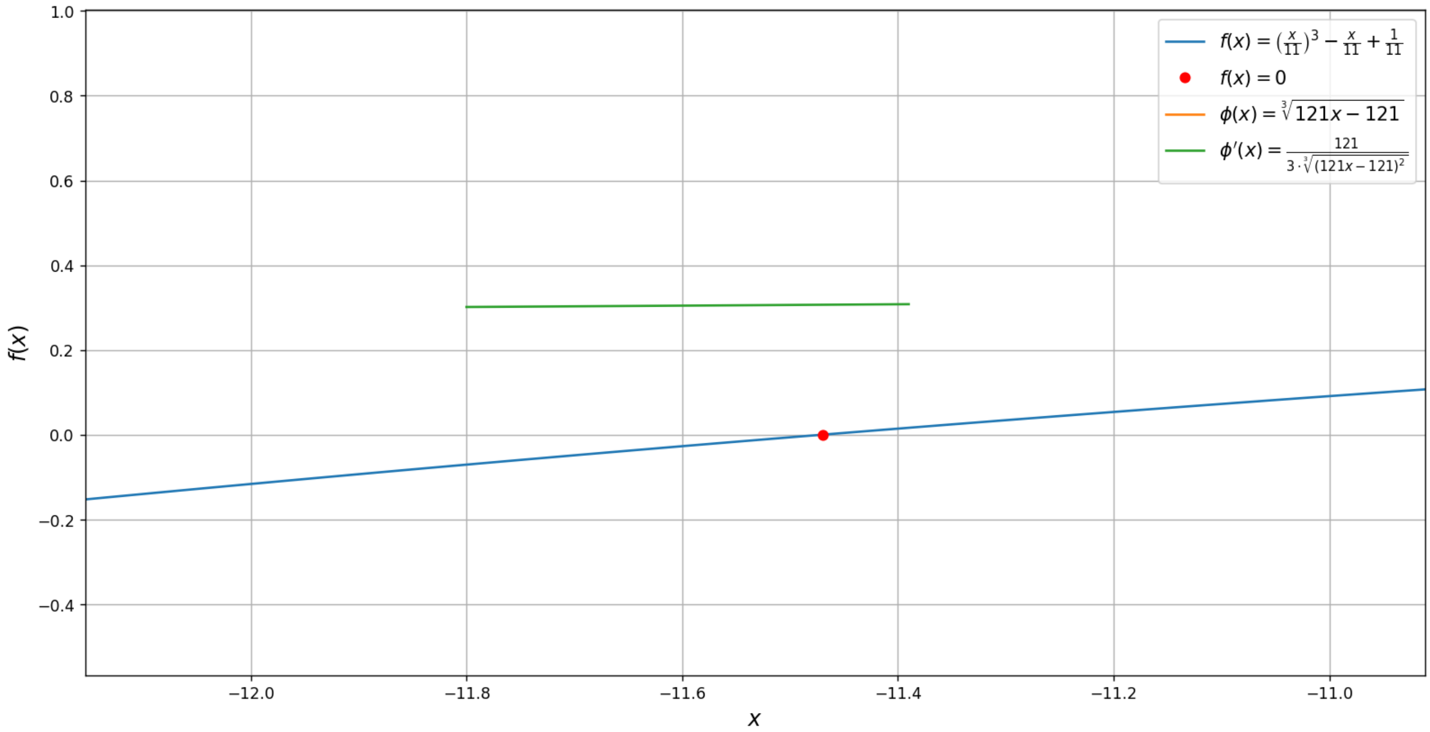


Рис. 10: Метод итераций

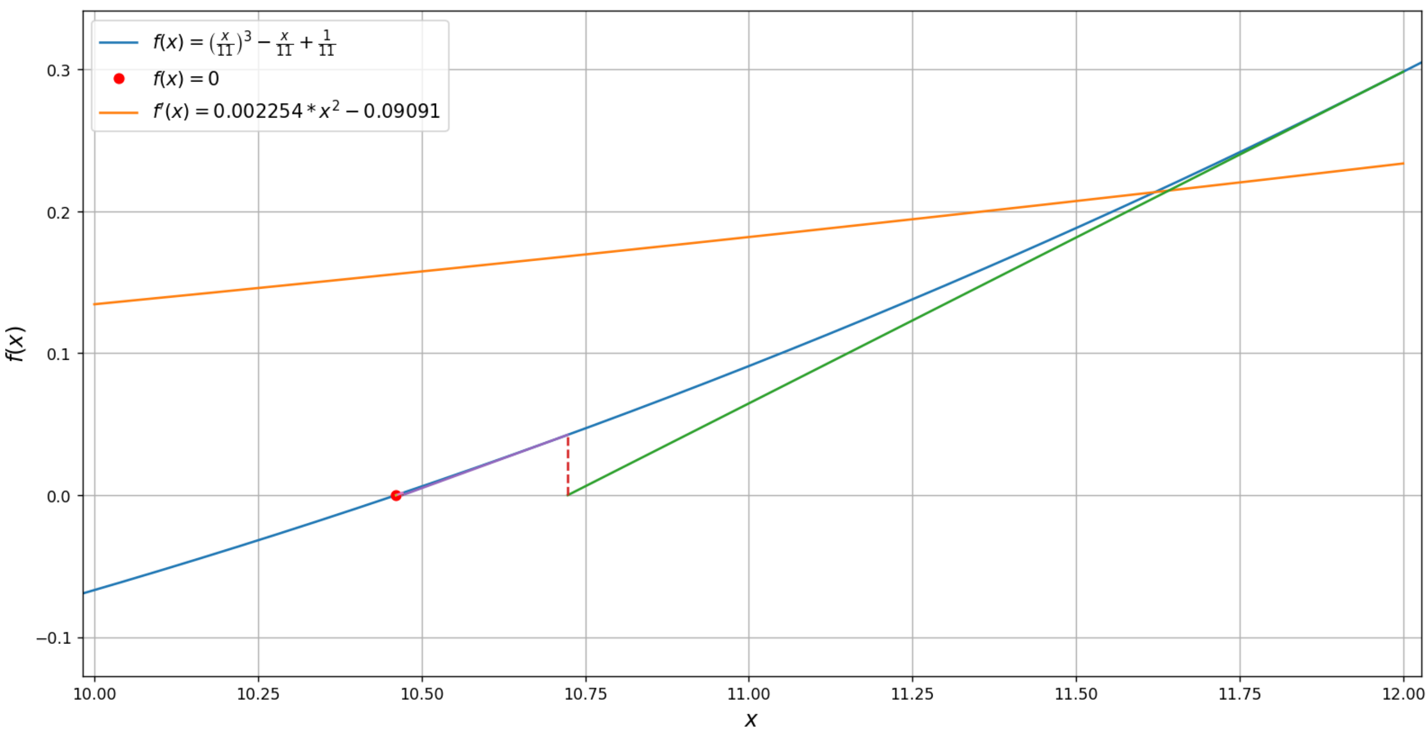


Рис. 11: Метод Ньютона

Результат

```
        Метод дихотомии:
a = -11.4697265625; b = -11.468505859375
f(x) = 0 в [-15; -10] при x = -11.4691162109375

        Метод итераций:
f(x) = 0 в [-11.8; -11.4] при x = -11.469863875464782

        Метод Ньютона (касательных):
f(x) = 0 в [10; 12] при x = 10.461035686795155
```

6 Задание №6

Условие

По заданной таблице построить интерполяционный многочлен:

X	-2	-1	0	1	2
Y	11	11	-1	11	11

а) по первым четырём узлам;

б) по последним четырём узлам;

в) по всем пяти узлам.

Алгоритм

Реализуем нахождение многочлена Лагранжа.

```
result = np.poly1d([])
for i in range(len(x)):
    result += y[i] * math.prod(np.poly1d([1, -x[j]]) / (x[i] - x[j]) for j in range(len(x)) if j != i)
for i in range(len(x)):
    if round(result(x[i]), 10) != y[i]:
        print(f"Проверка: Error [{result(x[i])} != {y[i]}]")
        break
else:
    print("Проверка: OK")
```

Реализуем нахождение многочлена Ньютона через разделённые разности. Вычисляем разделённые разности и заносим их в словарь *self.buffer*. Находим многочлен как сумму произведений разделённых разностей и разностей $(x - x_i)$.

```
def get_f(self, indexes):
    if len(indexes) == 1:
        return self.y[indexes[0] - 1]
    map_key = "".join(str(index) for index in indexes)
    if map_key in self.buffer.keys():
        return self.buffer.get(map_key)
    result = (self.get_f(indexes[1:]) - self.get_f(indexes[:-1])) / (self.x[indexes[-1] - 1] -
    ↪  self.x[indexes[0] - 1])
    self.buffer[map_key] = result
    return result

def get_coefficient(self, number):
```

```
    return self.buffer.get("".join(str(index) for index in range(1, int(number) + 1)))

def get_expression(self):
    self.get_f(np.arange(1, len(self.x) + 1, 1))
    result = np.poly1d([self.get_f([1])])
    result += sum(self.get_coefficient(i) * math.prod(np.poly1d([1, -self.x[j]]) for j in range(i - 1)) for i in range(2, len(self.x) + 1))
    return result
```

Результат

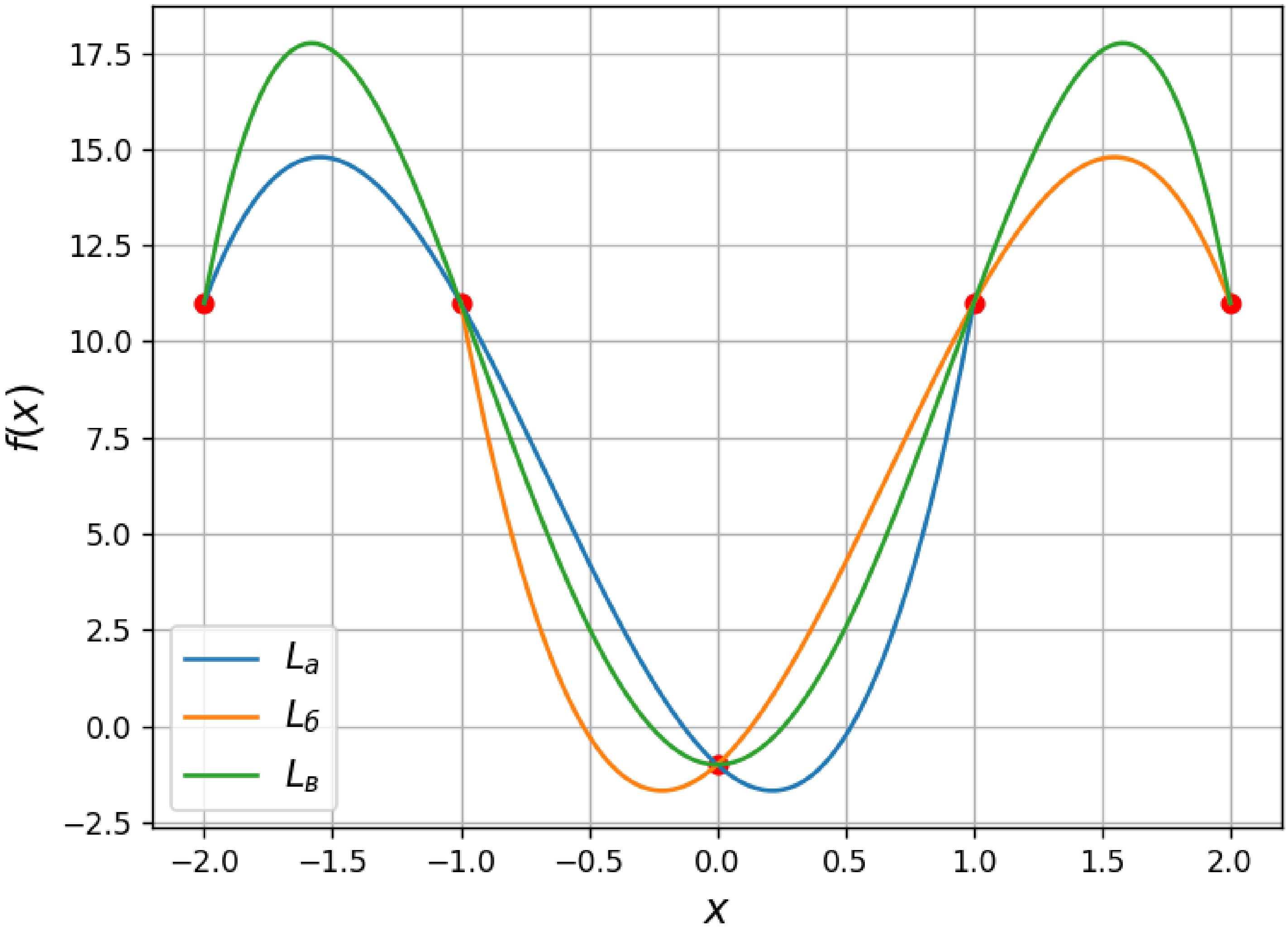


Рис. 12: Многочлен Лагранжа

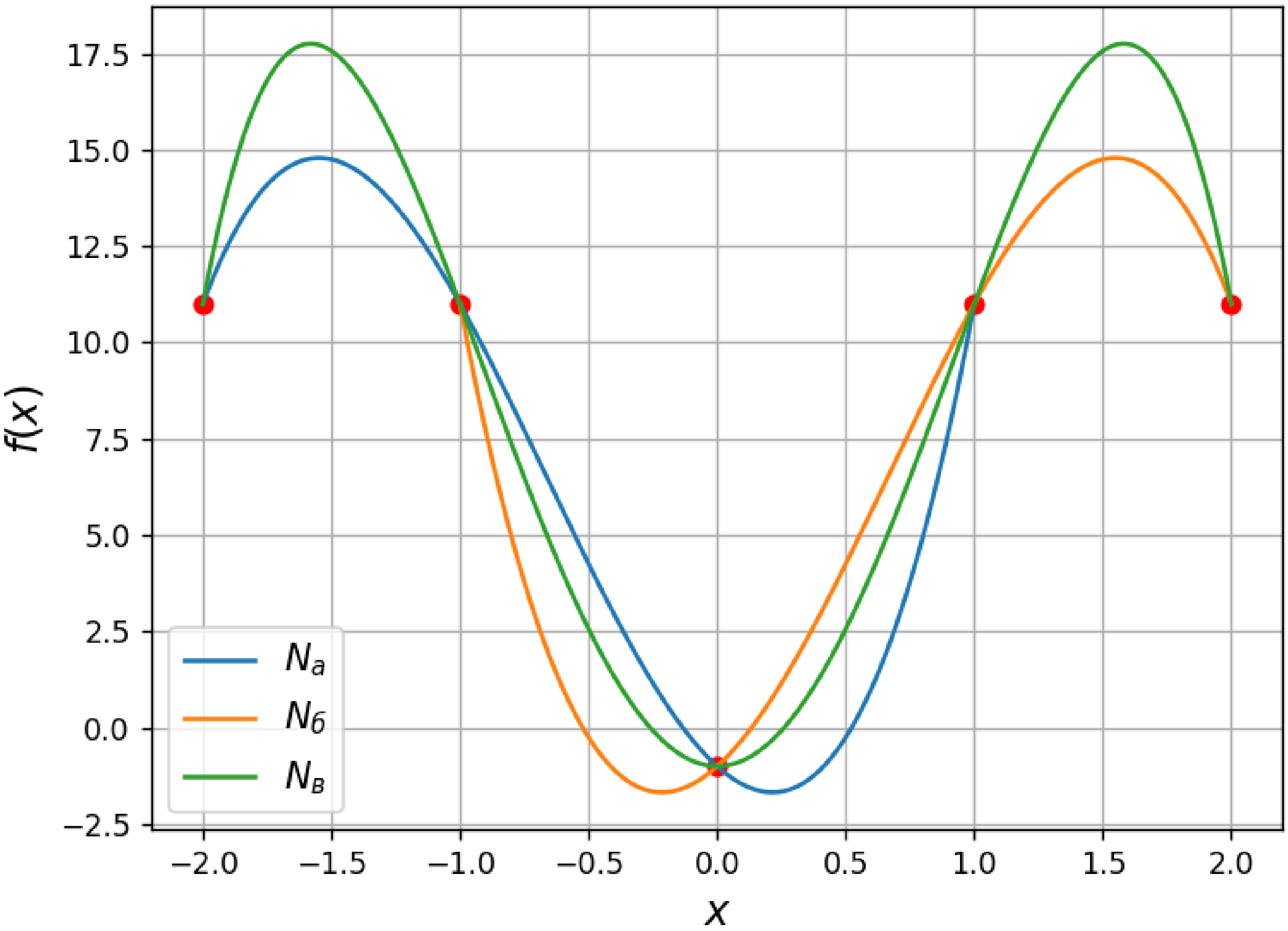


Рис. 13: Многочлен Ньютона

7 Задание №7

Условие

По заданной таблице методом наименьших квадратов построить многочлены первой и второй степени:

X	-55	-33	-11	11	33
Y	73	51	30	11	-8

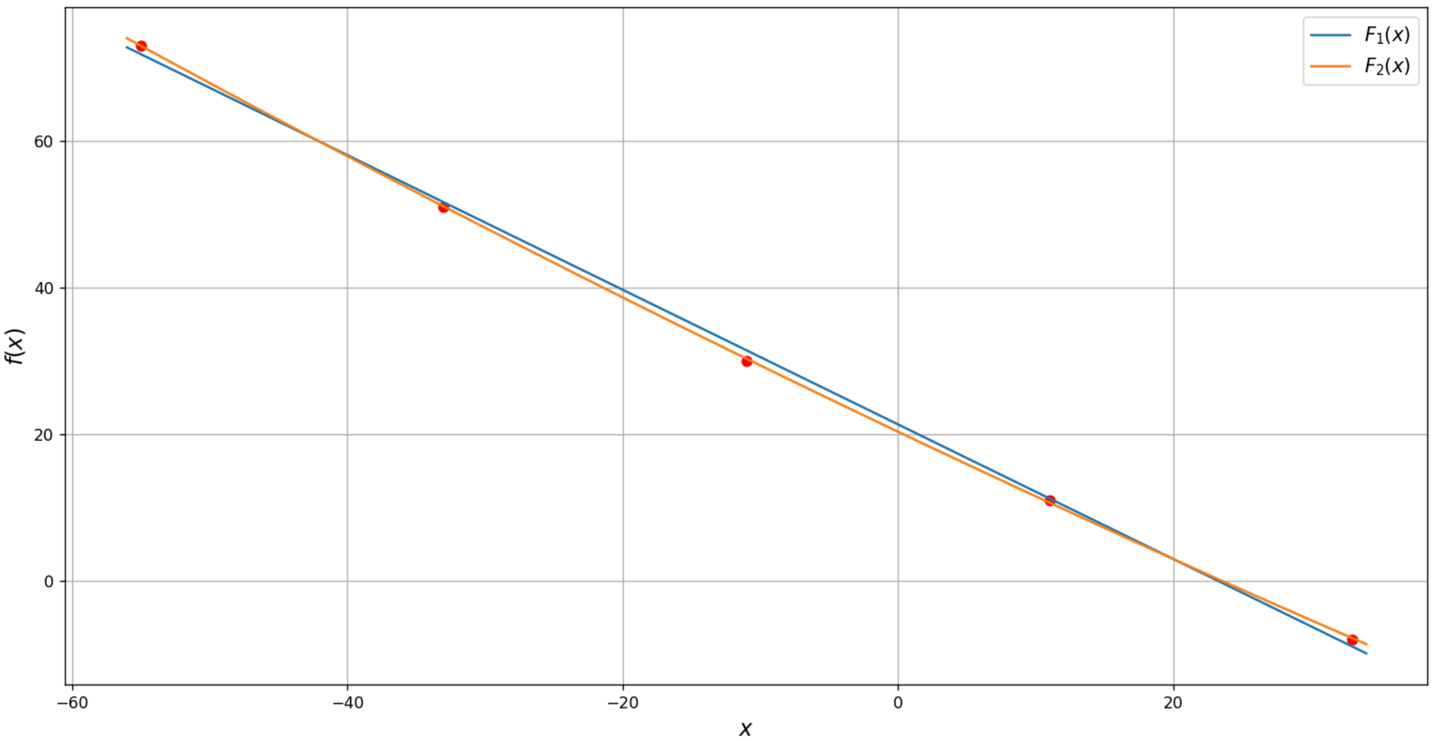
Алгоритм

Вычисляем коэффициенты системы с помощью метода *get_sum_x(...)*. Затем с помощью метода Гаусса получаем коэффициенты многочлена и вычисляем невязку.

```
def get_sum_x(x_list, degree):
    if not degree:
        return len(x_list)
    return sum(math.pow(x_value, degree) for x_value in x_list)

def get_F(x, y, degree):
    degree += 1
    a = [[0] * degree for i in range(degree)]
    for i in range(degree):
        for j in range(degree):
            a[i][j] = get_sum_x(x, j + i)
        a[i].append(sum(y[index] * math.pow(x[index], i) for index in range(len(x))))
    result = get_a_by_Gauss(degree, a)
    print(f'\n\t\tРезультат F_{degree - 1}(x):')
    for index in range(degree):
        print(f'a_{index + 1} = {round(result[index], 4)}')
    print(f'\tНевязка: {round(sum((sum(result[i] * x[j] ** i for i in range(degree)) - y[j]) ** 2 for j in range(len(x))), 4)}')
    return result
```

График



Результат

```
        Результат F_1(x):
a_1 = 21.3
a_2 = -0.9182
    Невязка: 4.8

        Результат F_2(x):
a_1 = 20.3
a_2 = -0.8922
a_3 = 0.0012
    Невязка: 0.2286
```

8 Задание №8

Условие

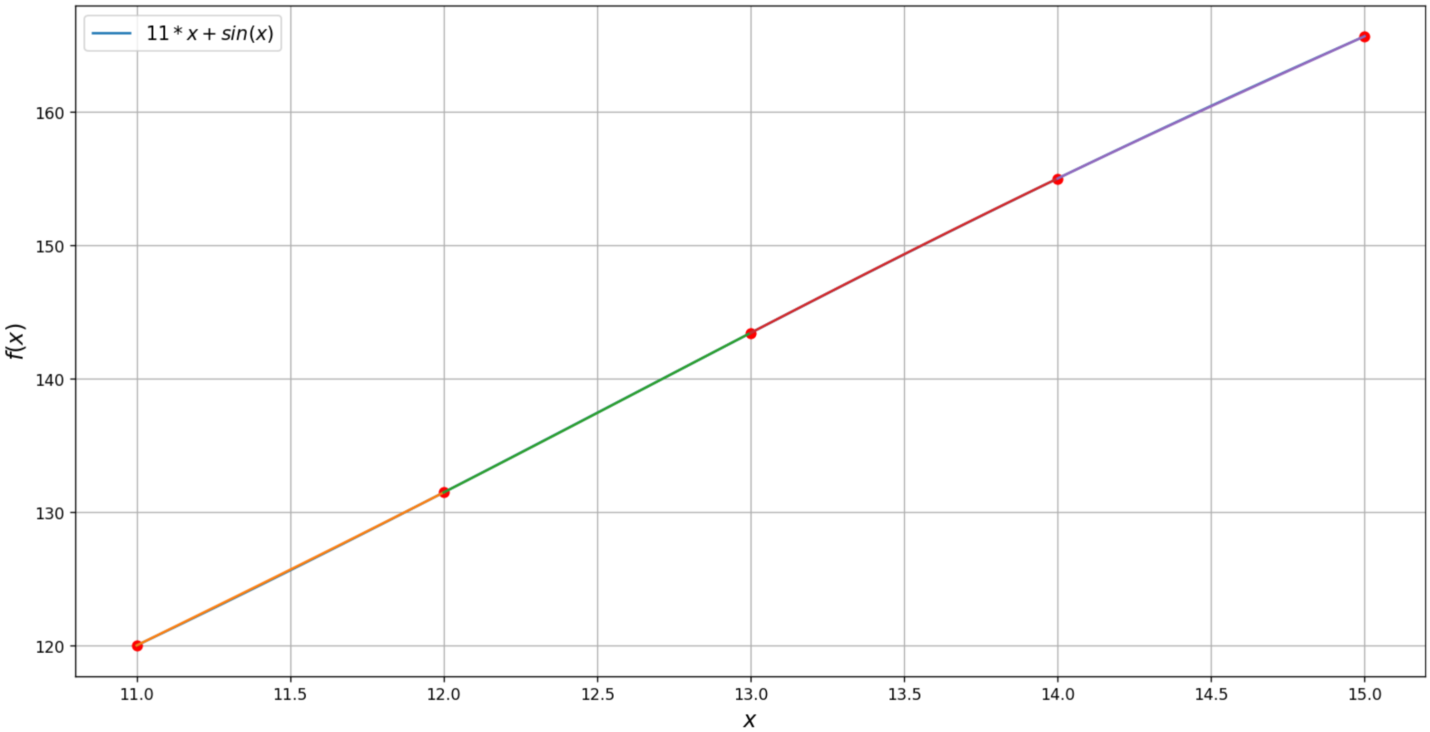
Аппроксимировать табличную функцию кубическими сплайнами. Переменная изменяется на отрезке $[a; b]$. Этот отрезок разбивается на N частей. Значения табличной функции на отрезке $[a; b]$ вычислять по формуле $f(x) = Nx + sin(x)$. $a = 11; b = 15; N = 4$. Нарисовать сплайн и табличную функцию.

Алгоритм

Вычисляем длины отрезков между табличными точками и составляем СЛАУ для поиска коэффициентов многочленов. Находим решение системы методом прогонки – функция *run_through(...)*.

```
def interpolation(x, y):
    h = [x[i + 1] - x[i] for i in range(len(x) - 1)]
    a = [0.0] + [h_element / 6 for h_element in h[1:-1]]
    b = [(h[i] + h[i + 1]) / 3 for i in range(len(h) - 1)]
    c = a[1:] + [0]
    d = [(y[i + 1] - y[i]) / h[i] - (y[i] - y[i - 1]) / h[i - 1] for i in range(1, len(h))]
    q = [0] + run_through(a, b, c, d) + [0]
    splines = dict()
    for i in range(1, len(q)):
        splines[x[i - 1]] = ((float(q[i]) * np.poly1d([1, -x[i - 1]]) ** 3 + float(q[i - 1]) *
        ↪   np.poly1d( [-1, x[i]]) ** 3) / 6 + (float(y[i]) - float(q[i]) * h[i - 1] ** 2 / 6) *
        ↪   np.poly1d([1, -x[i - 1]]) + (float(y[i - 1]) - float(q[i - 1]) * h[i - 1] ** 2 / 6) *
        ↪   np.poly1d([-1, x[i]])) / h[i - 1]
    return splines
```

Результат



$11 * x + sin(x)$
160
150
140
130
120
$f(x)$
11.0
11.5
12.0
12.5
13.0
13.5
14.0
14.5
15.0
x

9 Задание №9

Условие

Вычислить определённый интеграл $F = \int_{x_0}^{x_k} y\, dx$ методами прямоугольников, трапеций, Симпсона с шагами h_1, h_2. Уточнить полученные значения, используя метод Рунге-Ромберга.

$$y = \frac{1}{x^3 + 64}; \qquad X_0 = -2; \qquad X_k = 2; \qquad h_1 = 1; \qquad h_2 = 0.5$$

Алгоритм

Реализуем методы прямоугольников, трапеций и Симпсона.

```
def rectangle_method(x, h):
    return sum(h * function(x_value) for x_value in x[:-1])

def trapezoid_method(x, h):
    return h / 2 * (function(x[0]) + function(x[-1]) + 2 * sum(function(x[1:-1])))

def simpson_method(x, h):
    return h / 3 * (function(x[0]) + function(x[-1]) + 4 * sum(function(x[i]) for i in range(1, len(x),
    ↪ 2)) + 2 * sum(function(x[i]) for i in range(2, len(x) - 1, 2)))
```

Реализуем функцию *refine_value(...)* для уточнения значений с помощью метода Рунге-Ромберга.

```
def refine_value(full_value, half_value, p):
    return half_value + (half_value - full_value) / (2 ** p - 1)
```

Графики

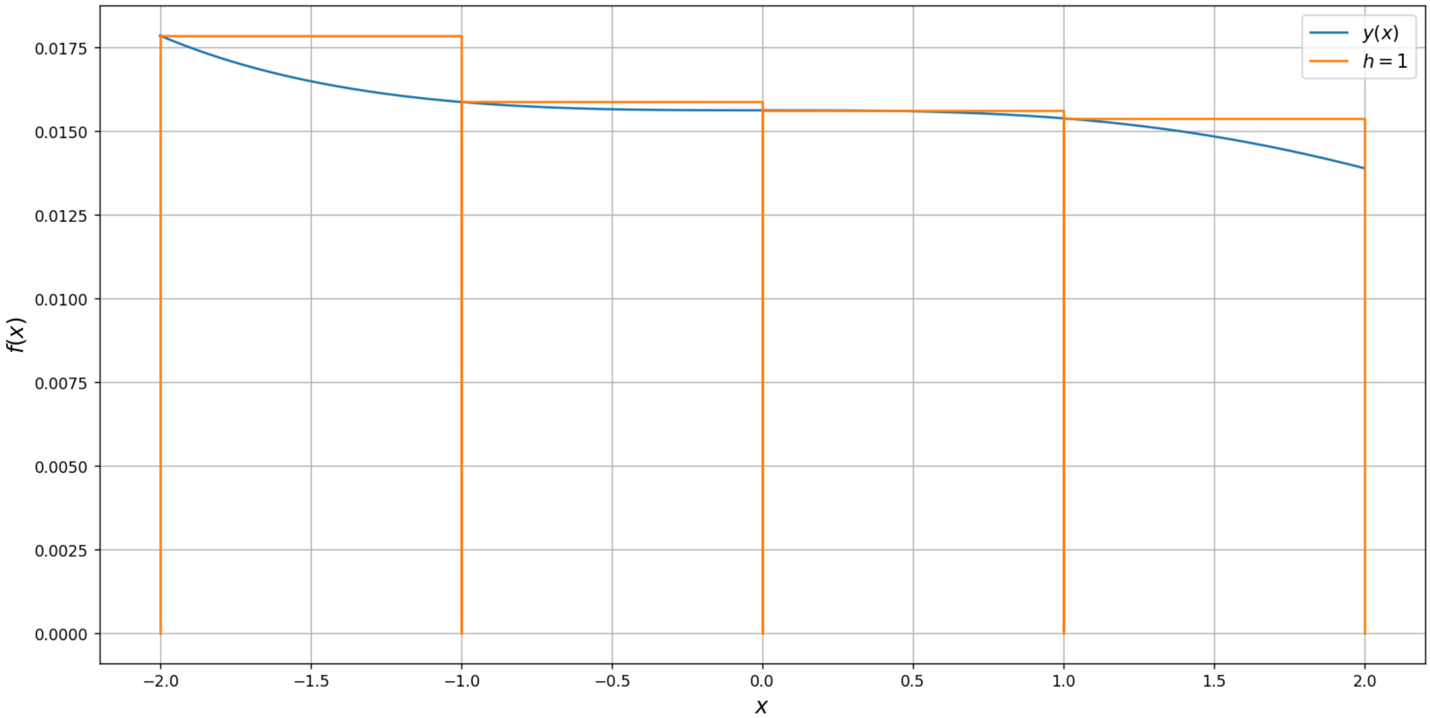


Рис. 14: Метод прямоугольников

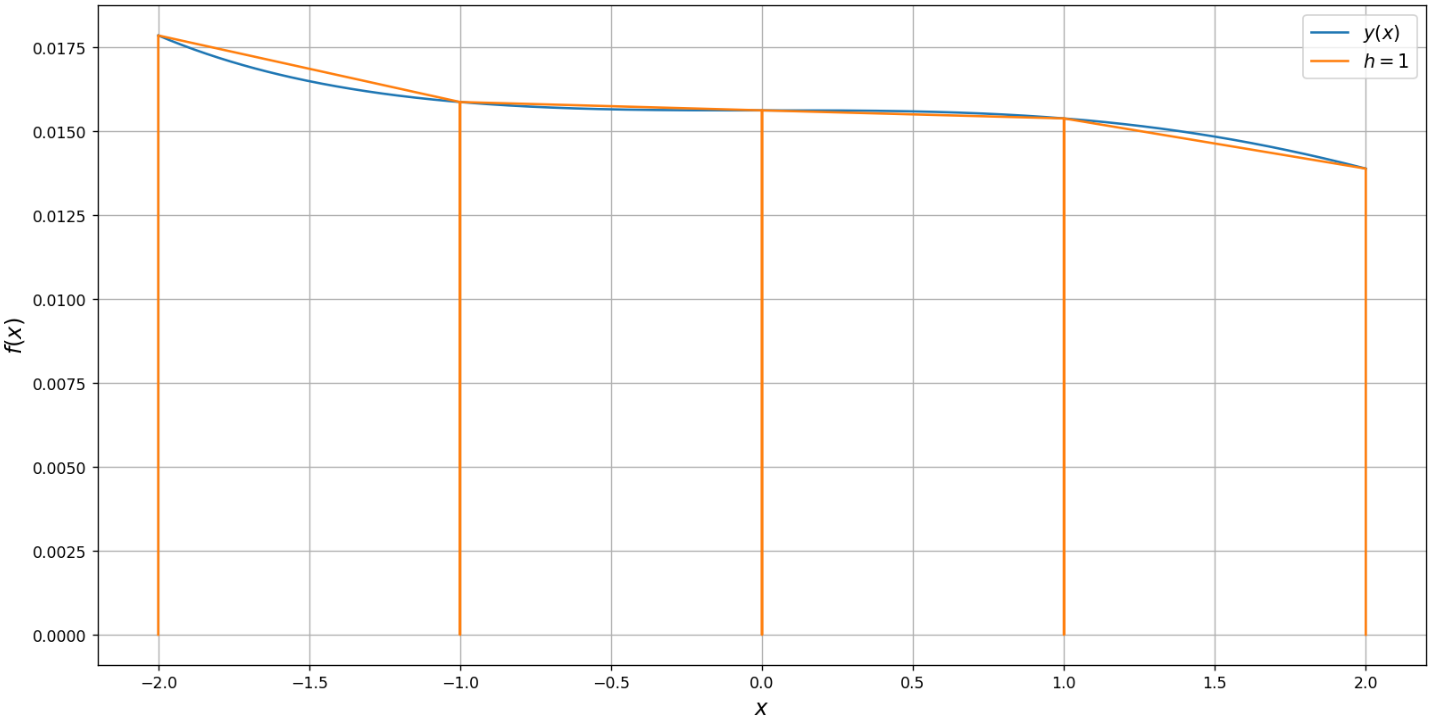


Рис. 15: Метод трапеций

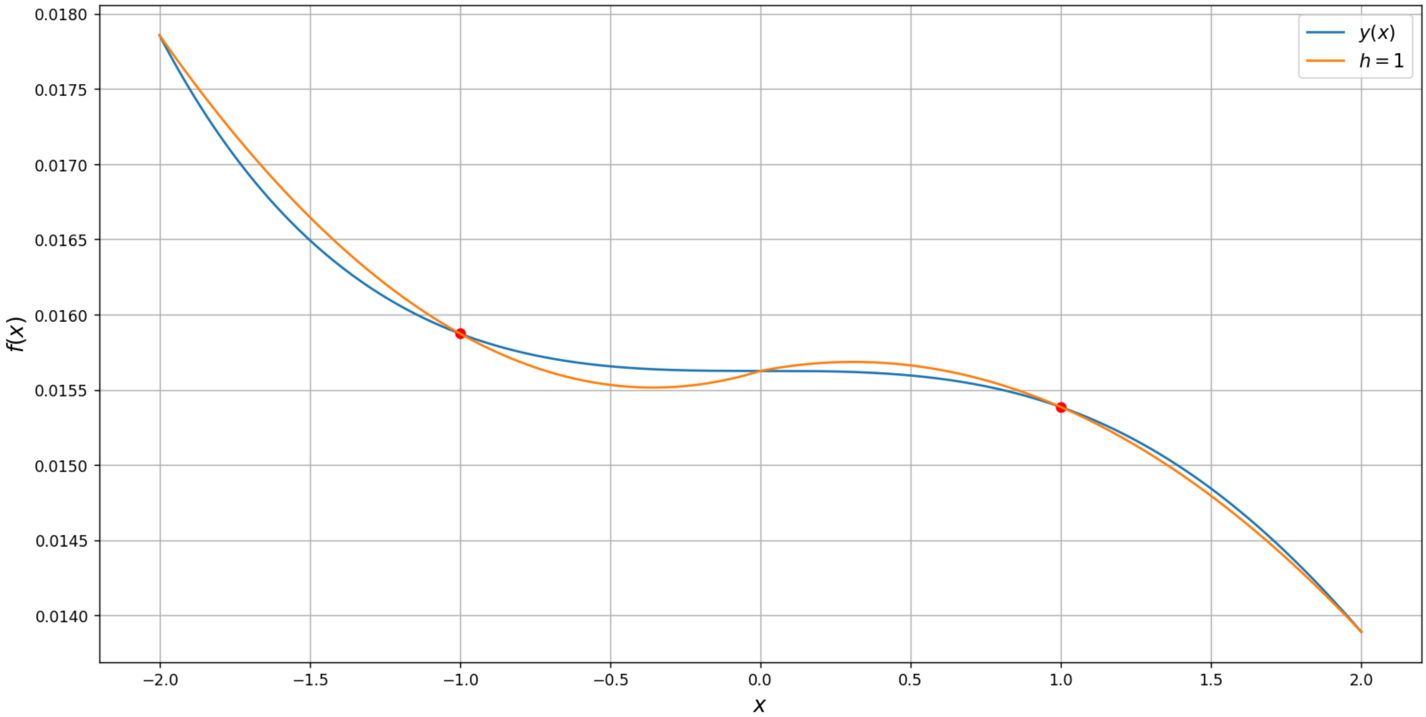


Рис. 16: Метод Симпсона

Результат

```
        Метод прямоугольников:
Значение при h = 1: 0.0647397741147741
Значение при h = 0.5: 0.06366351962096471
Уточнённое значение: 0.06258726512715532

        Метод трапеций:
Значение при h = 1: 0.06275564713064713
Значение при h = 0.5: 0.06267145612890121
Уточнённое значение: 0.06264339246165257

        Метод Симпсона:
Значение при h = 1: 0.06267551892551892
Значение при h = 0.5: 0.06264339246165257
Уточнённое значение: 0.06264125069739482
```

10 Задание №10

Условие

Решить задачу Коши для обыкновенного дифференциального уравнения 1-го порядка на указанном отрезке с заданным шагом h. Полученное численное решение сравнить с точным. Определить погрешность решения.

$$y' = 2\frac{x^2 - xy}{x^2 + 1}; \qquad y(1) = 0.6666667; \qquad x \in [1; 2]; \qquad h = 0.1;$$

Точное решение: $y = \frac{2}{3}\left(\frac{x^3+1}{x^2+1}\right)$.

Алгоритм

Реализуем методы Эйлера, Эйлера-Коши и Рунге-Кутта.

```python
def Euler(table: list, start_y, h):
    y = [start_y]
    for x_value in table[0][1:]:
        y.append(y[-1] + h * derivative(x_value, y[-1]))
    table.append(y)

def Euler_Cauchy(table: list, start_y, h):
    x = table[0]
    y = [start_y]
    for index in range(len(x) - 1):
        predictor = y[-1] + h * derivative(x[index], y[-1])
        y.append(y[-1] + h / 2 * (derivative(x[index], y[-1]) + derivative(x[index + 1], predictor)))
    table.append(y)

def Runge_Kutta(table: list, start_y, h):
    y = [start_y]
    for x_value in table[0]:
        k_1 = h * derivative(x_value, y[-1])
        k_2 = h * derivative(x_value + h / 2, y[-1] + k_1 / 2)
        k_3 = h * derivative(x_value + h / 2, y[-1] + k_2 / 2)
        k_4 = h * derivative(x_value + h, y[-1] + k_3)
        y.append(y[-1] + (k_1 + 2 * k_2 + 2 * k_3 + k_4) / 6)
    table.append(y)
```

Вычисляем погрешность решения как норму вектора разности точного решения и полученного.

```python
def inaccuracy(vector_1, vector_2):
    return max(abs(vector_1[i] - vector_2[i]) for i in range(1, len(vector_1)))
```

Результат

x:	1.0	1.1	1.8	1.9	2.0
y-точ.:	0.6666667	0.7031674	1.0742138	1.1365148	1.2
y-Эйл.:	0.6666667	0.7098039	1.1117084	1.1766868	1.2425519
y-Э-К:	0.6666667	0.7032428	1.0743795	1.1366727	1.2001493
y-Р-К:	0.6666667	0.7031675	1.074214	1.136515	1.2000001

```
Погрешность метода Эйлера: 0.04255189850484431
Погрешность метода Эйлера-Коши: 0.00017577714602345917
Погрешность метода Рунге-Кутта: 1.5029441291503076e-07
```

11 Список литературы

1. Черкасов М.А. Численные методы. Решение задач: Учебное пособие. М.:Изд-во МАИ, 2007. - 92С.:ил.

2. Пирумов У.Г. Численные методы: теория и практика. Учебное пособие для бакалавров. Издание 5, 2012 г.

3. Практические занятия в рамках учебного курса, преподаватель — Киндинова Виктория Валерьевна.

4. Лекционные занятия в рамках учебного курса, преподаватель — Гидаспов Владимир Юрьевич.