Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления »

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 6

Тема: Построение и программная реализация алгоритмов численного дифференцирования
Студент: Чаушев Александър

Группа: ИУ7-46Б

Оценка (баллы) _____

Преподаватель : Градов В. М.

Москва.
2020 г.

Цель работы: Получение навыков построения алгоритма вычисления производных от сеточных функций.

Входные данные: Задана табличная (сеточная) функция. Имеется информация, что закономерность, представленная этой таблицей, может быть описана формулой:

$$y = \frac{a_0 x}{a_1 + a_2 x}$$

x	y	1	2	3	4	5
1	0.571					
2	0.889					
3	1.091					
4	1.231					
5	1.333					
6	1.412					

Задача: Вычислить первые разностные производные от функции и занести их в столбцы 1- 4 таблицы. В столбец 5 занести вторую разностную производную.

Результат работы программы:

Заполненная таблица с краткими комментариями по поводу использованных формул и их точности.

Краткие теоретические сведения и алгоритм.

В основе численного дифференцирования лежит аппроксимация функции, от которой берется производная, интерполяционным многочленом. Все основные формулы численного дифференцирования могут быть получены при помощи первого интерполяционного многочлена Ньютона.

Полиномиальные формулы:

$$\varphi^{(k)}(x) = k!\left[y(x_0, x_1, \ldots, x_k) + \left(\sum_{i=0}^{k} \xi_i\right) y(x_0, x_1, \ldots, x_{k+1}) + \right.$$
$$+ \left(\sum_{i>j\geqslant 0}^{i=k+1} \xi_i \xi_j\right) y(x_0, x_1, \ldots, x_{k+2}) +$$
$$\left. + \left(\sum_{i>j>l\geqslant 0}^{i=k+2} \xi_i \xi_j \xi_l\right) y(x_0, x_1, \ldots, x_{k+3}) + \ldots\right].$$

Разложение в ряды Тейлора: Это наиболее универсальный метод построения формул численного дифференцирования заданных порядков точности относительно шага сетки.

$$y_i'' = \frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} + o(h^2)$$

Формулы Рунге:

Первая формула Рунге: при помощи можно оценить погрешность.

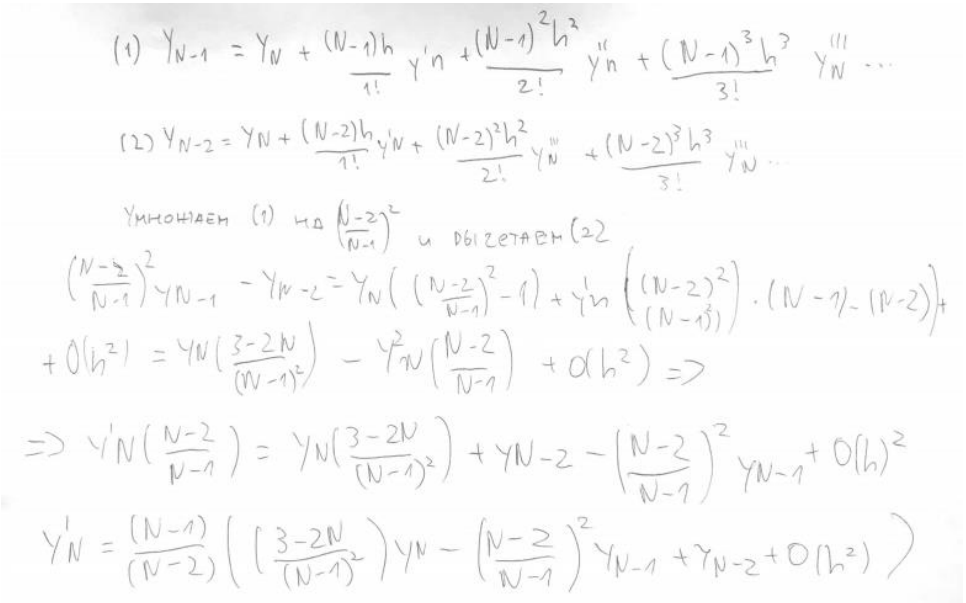
$$\psi(x)h^p = \frac{\Phi(h) - \Phi(mh)}{m^p - 1} + O(h^{p+1}).$$

Вторая формула Рунге позволяет за счет расчета на двух сетках с отличающимися шагами получить решение с более высокой точностью, чем заявленная теоретическая точность используемой формулы.

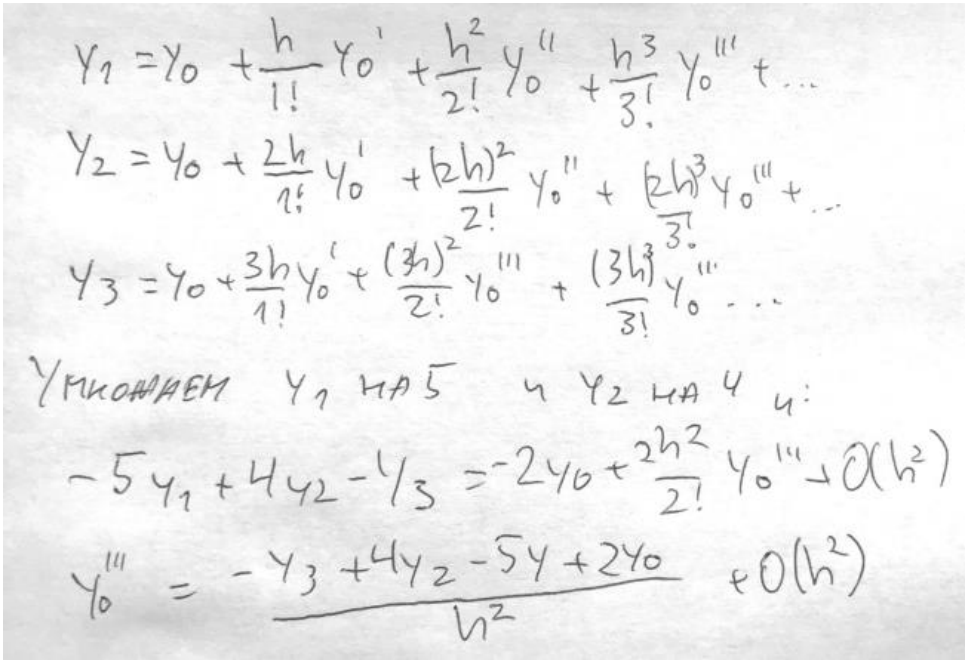
$$\Omega = \Phi(h) + \frac{\Phi(h) - \Phi(mh)}{m^p - 1} + O(h^{p+1}).$$

Ответы на контрольные вопросы:

1. Получить формулу порядка точности O(h^2) для первой разностной производной y’n в крайнем правом узле xn.

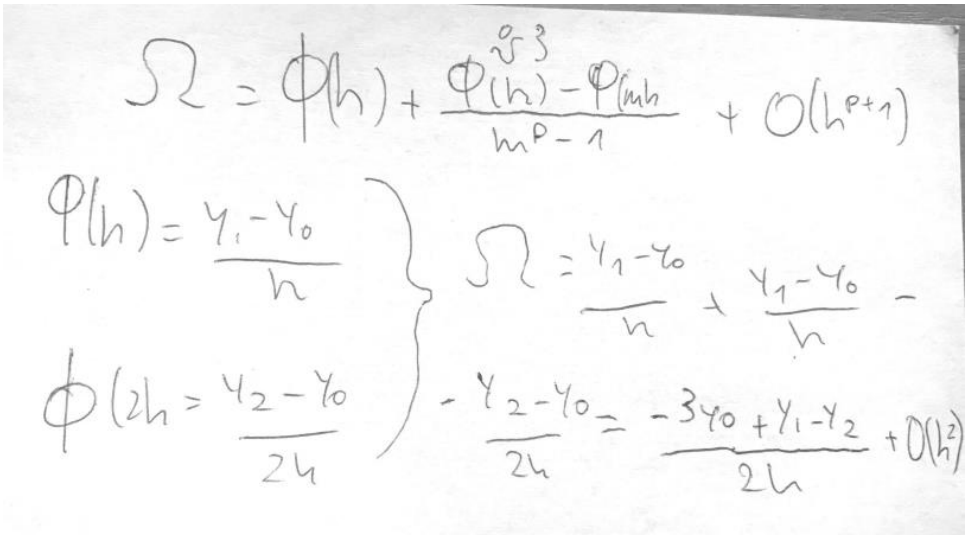


2. Получить формулу порядка точности O(h^2) для второй разностной производной y’’0 в крайнем левом узле x0.

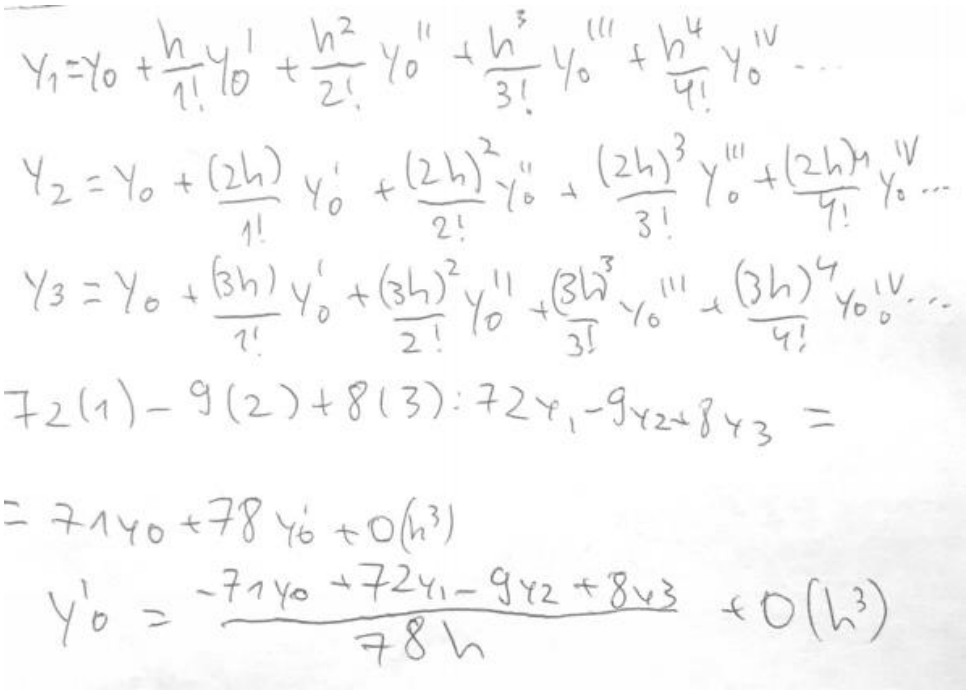


3. Используя 2-ую формулу Рунге, дать вывод выражения (9) из Лекции №7 для первой y'_0 производной в левом крайнем узле

$$y'_0 = \frac{-3y_0 + 4y_1 - y_2}{2h} + O(h^2)$$



4. Любым способом из Лекций №7, 8 получить формулу порядка точности O(h^3) для первой разностной производной y'0 в крайнем левом узле x0.



Код программы:

```
def f(x, a0 = 1, a1 = 2, a2 = 3):
   return a0 * x / (a1 + a2 * x)

def f_derivate(x, a0 = 1, a1 = 2, a2 = 3):
   return a0 * a1 / ((a1 + a2 * x) ** 2)


# ksi(x) = 1 / x
# eta(y) = 1 / y = (a1 + a2*x) / (a0*x) = a2/a0 + a1/a0 * 1/x = a + b*ksi
# eta'|y = 1 / (y * y)
# ksi'|y = 1 / (x * x)
# eta'|ksi = (a + b*ksi)' = b = a1/a0

def ksi(x, a0 = 1, a1 = 2):
   return (a1 / a0) * x

def eta_y_derivate(y):
   return 1 / (y * y)
def ksi_x_derivate(x):
   return 1 / (x * x)


def left_side_form(y, h):
   y_len = len(y)
```

```
    result = [None] * y_len
    for i in range(1, y_len):
        result[i] = (y[i] - y[i - 1]) / h
    return result

def right_side_form(y, h):
    y_len = len(y)
    result = [None] * y_len
    for i in range(0, y_len - 1):
        result[i] = (y[i + 1] - y[i]) / h
    return result

def central_form(y, h):
    y_len = len(y)
    result = [None] * y_len
    for i in range(1, len(y) - 1):
        result[i] = (y[i + 1] - y[i - 1]) / (2 * h)
    return result

def calculate_bound_results(y, h):
    y_len = len(y)
    result = [None] * y_len
    result[0]         = -(3 * y[0]         - 4 * y[1]         + y[2])         / (2 * h)
    result[y_len - 1] =  (3 * y[y_len - 1] - 4 * y[y_len - 2] + y[y_len - 3]) / (2 * h)
    return result

def runge_left_side(y, h):
    y_len = len(y)
    r = 2
    p = 1
    zn = r ** p - 1

    y_h = left_side_form(y, h)
    y_h[0] = (y[1] - y[0]) / h

    y_rh = [None] * y_len
    for i in range(2, y_len):
        y_rh[i] = (y[i] - y[i - 2]) / (r * h)
    for i in range(2):
        y_rh[i] = (y[i + 2] - y[i]) / (r * h)
    result = [None] * y_len
    for i in range(y_len):
        result[i] = y_h[i] + (y_h[i] - y_rh[i]) / zn
    return result

def align_veriable(x, y):
    y_len = len(y)
    result = [0] * y_len

    tmp = [ksi(x[i]) for i in range(1, len(x))]
```

```
    tmp = right_side_form(tmp, h)
    eta_ksi_derivate = tmp[0]

    for i in range(0, y_len):
        if x[i] == 0:
            result[i] = None
        else:
            result[i] = (ksi_x_derivate(x[i]) / eta_y_derivate(y[i])) * eta_ksi_derivate
    return result


def real_derivate(x):
    return [f_derivate(i) for i in x]

def fill_table(x_start, x_end, step):
    x_table = [float(i) for i in range(x_start, x_end, step)]
    y_table = [f(i) for i in x_table]
    return x_table, y_table

def calculate(x: list, y: list, h: float):
  res = [x, y]
  res.append(real_derivate(x))
  res.append(left_side_form(y, h))
  res.append(right_side_form(y,h))
  res.append(calculate_bound_results(y, h))
  res.append(central_form(y, h))
  res.append(runge_left_side(y, h))
  res.append(align_veriable(x, y))

  return res

def print_table(table):
  print("|{0:^13}|{1:^13}|{2:^13}|{3:^13}|"
        "{4:^13}|{5:^13}|{6:^13}|{7:^13}|"
        "{8:^16}|".format("x", "y","Real",
                          "Left Side", "Right Side",
                          "Bounds", "Central", "Runge",
                          "Align Variable"))
  # Change None to <->
  for row in table:
    for i in range(len(row)):
      if row[i] == None:
        row[i] = "<->"

  for i in range(len(table[0])):
    print( "|{0:^13.3}|{1:^13.3}|{2:^13.3}|{3:^13.3}|"
          "{4:^13.3}|{5:^13.3}|{6:^13.3}|{7:^13.3}|"
          "{8:^16.3}|".format(table[0][i], table[1][i],
                          table[2][i], table[3][i],
                          table[4][i], table[5][i],
```

```
                table[6][i], table[7][i],
                table[8][i]))

if __name__ == "__main__":
  h = 1.
  x, y = fill_table(1, 11, 1)
  res_table = calculate(x, y, h)
  print_table(res_table)
```

Результат работы

x	y	Real	Left Side	Right Side	Bounds	Central	Runge	Align Variable
1.0	0.2	0.08	<->	0.05	0.0636	<->	0.0636	0.08
2.0	0.25	0.0312	0.05	0.0227	<->	0.0364	0.0821	0.0312
3.0	0.273	0.0165	0.0227	0.013	<->	0.0179	0.00909	0.0165
4.0	0.286	0.0102	0.013	0.0084	<->	0.0107	0.00812	0.0102
5.0	0.294	0.00692	0.0084	0.00588	<->	0.00714	0.00611	0.00692
6.0	0.3	0.005	0.00588	0.00435	<->	0.00512	0.00462	0.005
7.0	0.304	0.00378	0.00435	0.00334	<->	0.00385	0.00358	0.00378
8.0	0.308	0.00296	0.00334	0.00265	<->	0.003	0.00284	0.00296
9.0	0.31	0.00238	0.00265	0.00216	<->	0.0024	0.00231	0.00238
10.0	0.312	0.00195	0.00216	<->	0.00191	<->	0.00191	0.00195