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 (ИУ-7)»

Лабораторная работа № 3

Тема Построение и программная реализация алгоритма сплайн-интерполяции табличных функций.

Студент Сироткина П.Ю.

Группа ИУ7-46Б

Оценка (баллы)

Преподаватель Градов В.М.

Москва, 2021 год

Цель работы: получение навыков владения методами интерполяции таблично заданных функций с помощью кубических сплайнов.

Исходные данные:

1. Таблица функции с количеством узлов N. Задается с помощью формулы y = x^2 в диапазоне [0...10] с шагом 1.

2. Значение аргумента x в первом интервале, например, при x = 0.5 и в середине таблицы, например, при x = 5.5. Сравнить с точным значением

Результат работы программы.

1. Значения y(x).

2. Сравнить результаты интерполяции кубическим сплайном и полиномом Ньютона 3-ей степени.

Описание алгоритма:

TODO

Код программы:

(Реализация метода Ньютона была предоставлена в прошлых лабораторных. Полный код предоставлен во вложении.)

Функция интерполирования сплайном:

```
def spline_interpolation(table, x):
    pos, side = get_the_nearest_dot(table, x)

    if side == "right":
        i = pos
    else:
        i = pos + 1

    a = table[i - 1][1]

    c = [0 for k in range(len(table) + 1)]
    ksi = [0 for k in range(len(table) + 1)]
    etta = [0 for k in range(len(table) + 1)]

    # h = 1 для всех i
```

(Продолжение)

```
    for k in range(2, len(table)):
        f = 3 * (table[k][1] - 2 * table[k - 1][1] + table[k - 2][1])

        ksi[k + 1] = -1 / (ksi[k] + 4)

        etta[k + 1] = (f - etta[k]) / (ksi[k] + 4)

    c[len(table) - 1] = etta[len(table)]

    for k in range(len(table) - 2, 0, -1):
        c[k] = ksi[k + 1] * c[k + 1] + etta[k + 1]

    b = table[i][1] - table[i - 1][1] - (c[i + 1] - 2 * c[i]) / 3

    d = (c[i + 1] - c[i]) / 3

    c = c[i]

    return a + b * (x - table[i - 1][0]) + c * (x - table[i - 1][0]) ** 2 + d * (x - table[i - 1][0]) ** 3
```

Функция вычисления ближайшей точки в интервале:

```python
def get_the_nearest_dot(table, x):
    i = 0
    n = len(table)
    while i < n and table[i][0] < x:
        i += 1
    if i == n:
        i -= 1
    if i > 0 and (x - table[i - 1][0]) < (table[i][0] - x):
        return i - 1, "left"
    else:
        return i, "right"
```

Вызов функций:

```python
table = make_table_of_squares(10, 1)
print_table(table)

x1 = 0.5 # Значение аргумента x в 1-ом интервале для проведения интерполяции
x2 = 5.5 # Значение аргумента x в 6-ом интервале для проведения интерполяции

print("Newton interpolation:")
print("X = ", x1, ": F(x) ~= ", "%.2f" % newton_interpolation(table, x1, 3), sep="")
print("X = ", x2, ": F(x) ~= ", "%.2f" % newton_interpolation(table, x2, 3), sep="")
print()
print("Spline interpolation:")
print("X = ", x1, ": F(x) ~= ", "%.2f" % spline_interpolation(table, x1), sep="")
print("X = ", x2, ": F(x) ~= ", "%.2f" % spline_interpolation(table, x2), sep="")
print()
print("True values:")
print("X = ", x1, ": F(x) = ", "%.2f" % (x1 * x1), sep="")
print("X = ", x2, ": F(x) = ", "%.2f" % (x2 * x2), sep="")
```

Результат работы программы:

```
Newton interpolation:
X = 0.5: F(x) ~= 0.25
X = 5.5: F(x) ~= 30.25

Spline interpolation:
X = 0.5: F(x) ~= 0.34
X = 5.5: F(x) ~= 30.92

True values:
X = 0.5: F(x) = 0.25
X = 5.5: F(x) = 30.25
```

Как можно заметить, метод интерполяции сплайнами работает менее эффективно, чем интерполяция Ньютона.

Ответы на контрольные вопросы:

1. Получить выражения для коэффициентов кубического сплайна, построенного на двух точках.

Пусть i - 1 - номер первой точки, i - номер второй точки.

1 <= i <= N, где N - количество узлов.

Полином:

$F(x) = A_i + B_i * (x - x_{i-1}) + C_i * (x - x_{i-1}) ^ 2 + D_i * (x - x_{i-1}) ^ 3$ (1)

Найдем коэффициенты A_i, B_i, C_i, D_i.

$f(x_{i-1}) = y_{i-1} = A_i$ (2)

Введем обозначение $h_i = x_i - x_{i-1}$

$f(x_i) = y_i = A_i + B_i * h_i + C_i * h_i ^ 2 + D_i * h_i ^ 3$ (3)

Число таких уравнений меньше числа неизвестных в два раза. Недостающие уравнения можно получить, приравняв во внутренних узлах первые и вторые производные, вычисляемые по коэффициентам на соседних участках:

$f'(x) = B_i + 2 * C_i * (x - x_{i-1}) + 3 * D_i * (x - x_{i-1}) ^ 2$, (4)

$x_{i-1} <= x <= x_i$

$F''(x) = 2 * C_i + 6 * D_i * (x - x_{i-1})$ (5)

$B_{i+1} = B_i + 2 * C_i * h_i + 3 * D_i * h_i ^ 2$ (6)

$C_{i+1} = C_i + 3 * D_i * h_i$ (7)

Также предполагают следующие условия:

$f''(x_0) = 0, C_1 = 0$ (8)

$f''(x_N) = 0, C_{N+1} = C_N + 3 * D_N * h_N = 0$ (9)

Из (1) находятся A_i.

Из (7) и (9) следует:

$D_i = (C_{i+1} - C_i) / (3 * h_i)$

$D_N = - C_N / (3 * h_N)$

Исключим из (3) D_i с помощью полученного выше выражения, получим:

$B = (y_i – y_{i - 1}) / h_i – h_i * (C_{i + 1} – 2 * C_i) / 3, \ 1 <= i < N.$

Из (3), используя выражение для D_N получим:

$B_N = (y_N – y_{N - 1}) / h_N – h_N * 2 * C_N / 3$

Теперь исключим из (6) величины B_i и $B_{i + 1}$ с учетом полученных выше выражений, а также величину D_i. В результате получим систему уравнений для определения C_i:

$C_1 = 0$

$h_{i - 1}* C_{i - 1} + 2 * (h_{i - 1} + h_i) * C_i + h_i * C_{i + 1} = 3 * ((y_i – y_{i - 1}) / h_i - (y_{i - 1} – y_{i - 2}) / h_i)$

$C_{N + 1} = 0$

После нахождения коэффициентов C_i находятся остальные коэффициенты по вычисленным ранее формулам, выраженным через $C_{i.}$

Заметим, что полученная система - СЛАУ с трехдиагональной матрицей. Она решается методом прогонки.

В каждом уравнении содержится только два неизвестных, и при обратном ходе одно из этих неизвестных выражается через другое.

$C_i = e_{i + 1} * C_{i + 1} + n_{i + 1}$, где $e_{i + 1}$ и $n_{i + 1}$ - пока неизвестные коэффициенты.

Можно записать так: $C_{i - 1} = e_i * C_i + n_i$

Подставляя полученное выражение во второе уравнение системы и преобразуя, получим:

$C_i = - h_i / (h_{i - 1} * e_i + 2 * (h_{i - 1} + h_i)) * C_{i + 1} + (f_i – h_{i - 1} * n_i) / (h_{i - 1} * e_i + 2 * (h_{i - 1} + h_i))$

Где $f = 3 * ((y_i – y_{i - 1}) / h_i - (y_{i - 1} – y_{i - 2}) / h_i)$

Сравнивая это выражение с выражением для C_i, получаем:

$e_{i + 1} = - h_i / (h_{i - 1} * e_i + 2 * (h_{i - 1} + h_i))$

$n_{i + 1} = (f_i – h_{i - 1} * n_i) / (h_{i - 1} * e_i + 2 * (h_{i - 1} + h_i))$

Из условия $C_1 = 0$ следует, что $e_2 = 0$ и $n_2 = 0$.

Теперь алгоритм решения системы выглядит следующим образом. По формулам для e_{i+1} и n_{i+1} при известных e_i и n_i вычисляют прогоночные коэффициенты (прямой ход). Затем по формуле $C_i = e_{i+1} * C_{i+1} + n_{i+1}$ и при условии $C_{N+1} = 0$ определяют все C_i (обратный ход).

2. Выписать все условия для определения коэффициентов сплайна, построенного на 3-х точках.

Необходимо определить 8 коэффициентов => 8 условий:

Функция должна пройти через точки, ограничивающие исследуемый отрезок - 4 условия.

Равенство производных первой и второй степени во внутренних узлах - 2 условия.

Равенство нулю производных функции на концах отрезка (вводимое допущение) - 2 условия.

3. Определить начальные значения прогоночных коэффициентов, если принять, что для коэффициентов сплайна справедливо C1=C2.

$C_1 = e_2 * C_2 + n_2$

$C_1 = C_2 \Rightarrow C_1 = e_2 * C_1 + n_2$

$e_2 * C_1 - C_1 + n_2 = 0$

$C_1 * (e_2 - 1) + n_2 = 0$

C_1 предполагается равным 0 по алгоритму решения:

$0 + n_2 = 0$

$n_2 = 0$

e_2 может быть любым.

Если же $C_1 \mathrel{!}= 0$, то необходимо выбрать $e_2 = 1$, $n_2 = 0$.

4. Написать формулу для определения последнего коэффициента сплайна CN, чтобы можно было выполнить обратный ход метода прогонки, если в качестве граничного условия задано kCN-1+mCN=p, где k,m и p - заданные числа.

$C_{N-1} = (-m / k) * C_N + (p / k)$

$e_N = (-m/k)$

$n_N = p / k$

$C_N = e_{N+1} * C_{N+1} + n_{N+1}$

$C_{N+1} = 0 \Rightarrow C_N = n_{N+1}$

$n_{N+1} = (f_N - n_N) / (e_N + 4)$ (примем $h_i = 1$ для удобства)

$n_{N+1} = (f_N - p / k) / ((-m/k) + 4)$