Федеральное государственное автономное образовательное
учреждение высшего образования
Национальный исследовательский
университет ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки:

Системное и Прикладное Программное Обеспечение

(09.03.04 Программная инженерия)

Дисциплина «Вычислительная математика»

Отчет

По лабораторной работе №5

Вариант №10

Студент

Карташев Владимир Сергеевич,
группа P3215

Преподаватель / Практик

Малышева Татьяна Алексеевна

г. Санкт-Петербург, 2024 г.

Цель работы

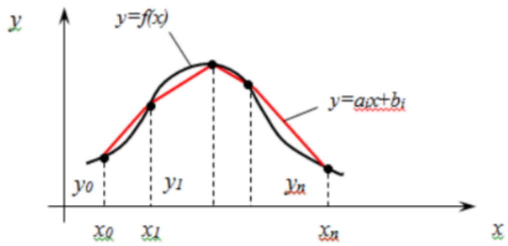
Решить задачу интерполяции, найти значения функции при заданных значениях аргумента, отличных от узловых точек.

Описание метода, расчетные формулы

Линейная интерполяция

Линейная интерполяция является простейшим и часто используемым видом локальной интерполяции . Она состоит в том, что заданные точки (x_i, y_i), соединяются прямолинейными отрезками, и функция $f(x)$ приближается к ломаной с вершинами в данных точках.

Уравнения каждого отрезка ломаной линии в общем случае разные. Поскольку имеется *n* интервалов (x_{i-1}, x_i) то для каждого из них в качестве уравнения интерполяционного полинома используется уравнение прямой, проходящей через две точки. В частности, для i — го интервала можно написать уравнение прямой, проходящей через точки (x_{i-1}, y_{i-1}) и (x_i, y_i), в виде: $\frac{y-y_{i-1}}{y_i-y_{i-1}} = \frac{x-x_{i-1}}{x_i-x_{i-1}}$



$$y = a_i x + b_i\ ,\ x_{i-1} \le x \le x_i \quad (2)$$

$$a_i = \frac{y_i-y_{i-1}}{x_i-x_{i-1}}$$

$$b_i = y_{i-1} - a_i x_{i-1}$$

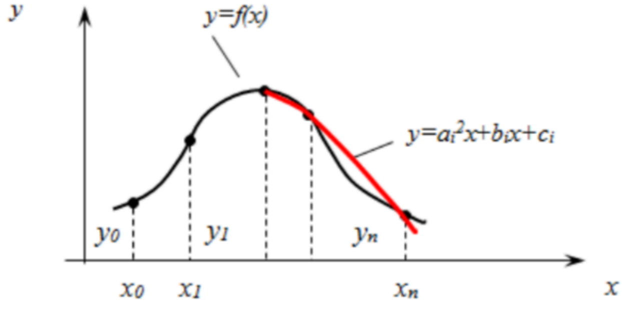
Следовательно, при использовании линейной интерполяции сначала нужно определить интервал, в который попадает значение аргумента *x*, а затем подставить его в формулу (2) и найти приближенное значение функций в этой точке.

Квадратичная интерполяция

В случае ***квадратичной интерполяции*** в качестве интерполяционной функции на отрезке $[x_{i-1},\ x_{i+1}]$ принимается квадратный трехчлен.

$$y = a_i x^2 + b_i x + c_i \qquad x_{i-1} \le x \le\ x_{i+1} \qquad (3)$$

Для определения неизвестных коэффициента a_i , b_i , c_i необходимы три уравнения. Ими служат условия прохождения параболы через три точки $(x_{i-1}, y_{i-1}), (x_i, y_i), (x_{i+1}, y_{i+1})$. Эти условия можно записать в виде:



$$a_i x_{i-1}^2 + b_i x_{i-1} + c_i = y_{i-1}$$

$$a_i x_i^2 + b_i x_i + c_i = y_i$$

$$a_i x_{i+1}^2 + b_i x_{i+1} + c_i = y_{i+1}$$

Интерполяция для любой точки $x \in [x_0,\ x_n]$ проводится по трем ближайшим ней узлам.

Интерполяция заданной функции

X	Y
2.1	3.7587
2.15	4.1861
2.2	4.9218
2.25	5.3487
2.3	5.9275
2.35	6.4193
2.4	7.0839

Таблица конечных разностей

№	x_i	y_i	Δy_i	$\Delta^2 y_i$	$\Delta^3 y_i$	$\Delta^4 y_i$	$\Delta^5 y_i$	$\Delta^6 y_i$
1	2.1	3.7587	0.4274	0.3083	-0.6171	1.0778	-1.7774	2.9757
2	2.15	4.1861	0.7357	-0.3088	0.4607	-0.6996	1.1983	
3	2.2	4.9218	0.4269	0.1519	-0.2389	0.4987		
4	2.25	5.3487	0.5788	-0.087	0.2598			
5	2.3	5.9275	0.4918	0.1728				
6	2.35	6.4193	0.6646					
7	2.4	7.0839						

Вычисление значения функции в X1

X1=2.355

Так как точка в правой половине отрезка интерполирования, то применим вторую интерполяционную формулу Ньютона.

$$t = \frac{x - x_n}{h} = \frac{2.355 - 2.4}{0.05} = \ - 0.9$$

y(X1) = N7 (t) = ys[6] + t*dys[5] + ((t*(t+1))/fct(2))*d2ys[4] + ((t*(t+1)*(t+2))/fct(3))*d3ys[3] + ((t*(t+1)*(t+2)*(t+3))/fct(4))*d4ys[2] + ((t*(t+1)*(t+2)*(t+3)*(t+4))/fct(5))*d5ys[1] + ((t*(t+1)*(t+2)*(t+3)*(t+4)*(t+5))/fct(6))*d6ys[0] = 6.452

Вычисление значения функции в X2

X2=2.254

a=2.25

Так как X2>a, то воспользуемся первой интерполяционной формулой Гаусса.

$$t = \frac{x - a}{h} = \frac{2.254 - 2.25}{0.05} = 0.08$$

y(X2) = P7 (t) = ys[3] + t*dys[3] + ((t*(t-1))/fct(2))*d2ys[2] + ((t*(t+1)*(t-1))/fct(3))*d3ys[2] + ((t*(t+1)*(t-1)*(t-2))/fct(4))*d4ys[1] + ((t*(t+1)*(t-1)*(t-2)*(t+2))/fct(5))*d5ys[1] + ((t*(t+1)*(t-1)*(t-2)*(t+2)*(t-3))/fct(6))*d6ys[0] = 5.3875

Листинг программы

Многочлен Лагранжа:

```
func l(x float64, xValues, yValues []float64) (l float64) {
   var numerator float64
   var denominator float64
   for i := 0; i < len(xValues); i++ {
      numerator = 1
      denominator = 1
      for j := 0; j < len(xValues); j++ {
         if i != j {
            numerator *= x - xValues[j]
            denominator *= xValues[i] - xValues[j]
         }
      }
      l += yValues[i] * numerator / denominator
   }
   return utils.Rounding(l)
}

func WithX(x float64, xValues, yValues []float64) {
   l := l(x, xValues, yValues)

   fmt.Println("l:", l)
}
```

Многочлен Ньютона с разделенными разностями:

```
func comp(y1, y0, x1, x0 float64) float64 {
   return (y1 - y0) / (x1 - x0)
}

func f(x float64, xValues, yValues []float64) float64 {
   max := len(xValues)

   var prev []float64 = yValues
   var curr []float64
   var fValues []float64
   var iteration int = 1
   for {
      if max == 1 {
         break
      }
      if iteration == 1 {
         for i := 0; i < max-1; i++ {
            curr = append(curr, comp(prev[i+1], prev[i], xValues[i+1], xValues[i]))
         }
         fValues = append(fValues, curr[0])
      } else {
         for i := 0; i < max-1; i++ {
            curr = append(curr, comp(prev[i+1], prev[i], xValues[iteration], xValues[0]))
         }
         fValues = append(fValues, curr[0])
      }
      //fmt.Println(curr)
      prev = curr
      fmt.Println(curr)
      curr = curr[:0]
      max--
      iteration++
   }

   //fmt.Println(fValues)

   var diffs []float64
```

```
    var mul float64 = 1
    for i := 0; i < len(xValues); i++ {
        mul *= x - xValues[i]
        diffs = append(diffs, mul)
    }

    //fmt.Println("diffs", diffs)
    var y float64 = yValues[0]

    for i := 0; i < len(fValues); i++ {
        y += fValues[i] * diffs[i]
        //fmt.Println(y)
    }

    fmt.Println(diffs)

    return y
}
```

Многочлен Ньютона с конечными разностями - нахождение приближенных значений:

```
func TransposeMatrix(matrix [][]float64) [][]float64 {
    // Проверяем, что матрица не пустая
    if len(matrix) == 0 {
        fmt.Println("Ошибка: матрица пустая")
        return nil
    }

    // Определяем количество строк и столбцов
    rows := len(matrix)
    cols := len(matrix[0])

    // Создаем новую матрицу для результата
    transposed := make([][]float64, cols)
    for i := range transposed {
        transposed[i] = make([]float64, rows)
    }

    // Транспонируем матрицу
    for i := 0; i < rows; i++ {
        for j := 0; j < cols; j++ {
            transposed[j][i] = matrix[i][j]
        }
    }

    return transposed
}

func Diffs(xValues, yValues []float64) (matrix [][]float64) {
    matrix = append(matrix, yValues)

    count := len(xValues) - 1
    var (
        curr []float64
        prev []float64 = yValues
    )

    for count > 0 {
        for i := 0; i < count; i++ {
            for j := count; j > 0; j-- {
                if j == 0 {
                    break
                }
```

```go
            curr = append(curr, utils.Rounding(prev[j]-prev[j-1]))
         }
         prev = utils.ReverseSlice(curr)

         for len(prev) < len(yValues) {
            prev = append(prev, 0)
         }

         matrix = append(matrix, prev)
         curr = curr[:0]
         count--
      }
   }

   matrix = TransposeMatrix(matrix)

   return
}

func interpolateForward(x float64, xValues []float64, diff [][]float64) (y float64) {
   fmt.Println("X:", x)

   order := 0
   for i := 0; i < len(xValues); i++ {
      if xValues[i] > x {
         order = i - 1
         break
      }
   }

   n := len(xValues) - (order + 1)
   fmt.Println("N:", n)

   //fmt.Println("Порядок:", order)

   t := utils.Rounding((x - xValues[order]) / utils.FindMiddleStep(xValues))
   fmt.Println("t:", t)

   var sub []float64
   var mul float64 = 1
   for i := 1; i < len(xValues)-(order+1); i++ {
      mul *= t - float64(i)
      sub = append(sub, utils.Rounding(mul))
   }

   y += diff[order][0]
   if len(xValues) == 1 {
      return y
   }
   //fmt.Println("y:", y)
   y += t * diff[order][1]
   if len(xValues) == 2 {
      return y
   }
   //fmt.Println("y:", y)

   count := 0
   for i := 2; i < len(diff[order])-order; i++ {
      y += ((t * sub[count]) / float64(utils.Factorial(i))) * diff[order][i]
      //fmt.Println("y:", y)
      count++
   }

   fmt.Println("Ответ:", y)

   r := math.Abs(t*sub[len(sub)-1]/float64(utils.Factorial(n+1))) * diff[0][n]
   fmt.Println("Погрешность интерполяции:", r)
```

```go
    return y
}

func interpolateBackward(x float64, xValues []float64, matrix [][]float64) (y float64) {
    fmt.Println("X:", x)

    order := 0
    for i := 0; i < len(xValues); i++ {
        if xValues[i] > x {
            order = i
            break
        }
    }

    n := order
    fmt.Println("N:", n)
    t := utils.Rounding((x - xValues[order]) / utils.FindMiddleStep(xValues))
    fmt.Println("t:", t)

    order = int(math.Abs(float64(order - 4)))

    var diff []float64
    rows := len(matrix)
    cols := len(matrix[0])

    // Перебираем элементы по диагонали
    for i := 0; i < rows && i < cols; i++ {
        diff = append(diff, matrix[rows-i-1][i])
    }

    var sub []float64
    var mul float64 = 1
    for i := 1; i < len(xValues)-(order+1); i++ {
        mul *= t + float64(i)
        sub = append(sub, utils.Rounding(mul))
    }

    y += diff[0]
    if len(xValues) == 1 {
        return y
    }
    y += t * diff[1]
    if len(xValues) == 2 {
        return y
    }

    count := 0
    for i := 2; i < len(diff); i++ {
        y += ((t * sub[count]) / float64(utils.Factorial(i))) * diff[i]
        count++
    }

    fmt.Println("Ответ:", y)

    r := math.Abs(t*sub[len(sub)-1]/float64(utils.Factorial(n+1))) * diff[n-1]
    fmt.Println("Погрешность интерполяции:", r)

    return y
}

func nOfFindingValues(findingValues []float64, xValues []float64, diff [][]float64) (y
[]float64) {
    middle := utils.FindMiddle(xValues)

    for _, v := range findingValues {
        fmt.Println()
        fmt.Println()
        if v <= middle {
```

```
            y = append(y, interpolateForward(v, xValues, diff))
        } else {
            y = append(y, interpolateBackward(v, xValues, diff))
        }
    }

    fmt.Println()
    return
}

func GetNewtonCoefficients(xValues, yValues []float64) (coefficients []float64) {
    diff := Diffs(xValues, yValues)

    step := utils.Rounding(utils.FindMiddleStep(xValues))

    sub := utils.CalculateCoefficientsForNewton(xValues)

    // Находим длину самой длинной строки в массиве
    maxLength := 0
    for _, row := range sub {
        if len(row) > maxLength {
            maxLength = len(row)
        }
    }

    // нули
    for i := range sub {
        for len(sub[i]) < maxLength {
            sub[i] = append([]float64{0}, sub[i]...)
        }
    }

    //fmt.Println("sub", sub)

    var tmpCoeff []float64
    diffY0 := diff[0]
    for i := 1; i < len(diffY0); i++ {
        if i == 0 {
            tmpCoeff = append(tmpCoeff, diffY0[i])
            continue
        }
        tmpCoeff = append(tmpCoeff,
utils.Rounding(diffY0[i]/(float64(utils.Factorial(i))*math.Pow(step, float64(i)))))
    }

    // добавляем все кроме первого
    tmpMatrix := utils.MultiplyMatrix(sub, tmpCoeff)
    coefficients = utils.SumColumns(tmpMatrix)
    //fmt.Println("coefficients:", tmpMatrix)

    //добавляем первый
    coefficients[len(coefficients)-1] = utils.Rounding(coefficients[len(coefficients)-1] +
diffY0[0])

    for i := range coefficients {
        coefficients[i] = utils.Rounding(coefficients[i])
    }

    return
}

func simpleN(xValues []float64, diff [][]float64) (coefficients []float64) {
    //order := len(xValues) - 1
    step := utils.FindMiddleStep(xValues)

    sub := utils.CalculateCoefficientsForNewton(xValues)

    // Находим длину самой длинной строки в массиве
```

```go
	maxLength := 0
	for _, row := range sub {
		if len(row) > maxLength {
			maxLength = len(row)
		}
	}

	// Добавляем нули перед элементами в строках, которые короче самой длинной строки
	for i := range sub {
		for len(sub[i]) < maxLength {
			sub[i] = append([]float64{0}, sub[i]...)
		}
	}

	var tmpCoeff []float64
	for i := 0; i < len(sub); i++ {
		tmpCoeff = append(tmpCoeff,
utils.Rounding(diff[0][i+1]/(float64(utils.Factorial(i+1))*math.Pow(step, float64(i+1)))))
	}
	// добавляем все кроме первого
	coefficients = utils.MulCoeffOnArray(tmpCoeff[0], sub[0])
	for i := 0; i < len(sub)-1; i++ {
		coefficients = utils.SumArrays(coefficients, utils.MulCoeffOnArray(tmpCoeff[i+1],
sub[i+1]))
		//fmt.Println(coefficients)
	}

	//добавляем первый
	for i := 0; i < len(coefficients); i++ {
		if i == len(coefficients)-1 {
			coefficients[i] = utils.Rounding(coefficients[i] - diff[0][0])
		}
	}

	return coefficients
}

func NewtonFiniteDifferencesWithoutFindingValues(xValues, yValues []float64) {
	diff := Diffs(xValues, yValues)
	printTableOfDiffs(xValues, diff)
	coefficients := simpleN(xValues, diff)
	fmt.Println(coefficients)
}
```

Примеры и результаты работы программы

Пример входных данных:

файл с названием input:

```
X=0.15
X 0.1 0.2 0.3 0.4 0.5
Y 1.25 2.38 3.79 5.44 7.14
```

Работа программы:

```
Выберите метод:
1. Многочлен Лагранжа
2. Многочлен Ньютона с разделенными разностями
3. Многочлен Ньютона с конечными разностями - нахождение приближенных значений
4. Многочлен Ньютона с конечными разностями - построение многочлена Ньютона по X и Y
Выберите тип ввода (1-3):
1
Выбран метод - Многочлен Лагранжа

Выберите тип ввода:
1. Ввод из файла
2. Ввод с консоли
Выберите тип ввода (1-2):
1

X: 0.15
X значения: [0.1 0.2 0.3 0.4 0.5]
Y значения: [1.25 2.38 3.79 5.44 7.14]

Степень искомого многочлена: 4
l: 1.7834

Лагранж:
-62.5001*x^4 + 55.8335*x^3 + -3.8751*x^2 + 9.4917*x^1 + 0.29
Ньютон:
-62.5*x^4 + 55.8333*x^3 + -3.875*x^2 + 9.4917*x^1 + 0.29


______
Выбран метод - Многочлен Ньютона с разделенными разностями

X: 0.15
X значения: [0.1 0.2 0.3 0.4 0.5]
Y значения: [1.25 2.38 3.79 5.44 7.14]
[11.299999999999999 14.100000000000005 16.499999999999996 16.999999999999996]
[14.000000000000032 11.99999999999996 2.5]
[-6.6666666666669085 -31.666666666666526]
[-62.49999999999904]
[0.04999999999999999 -0.0025000000000000005 0.00037500000000000006 -9.375000000000002e-05
3.2812500000000005e-05]
y(0.15) = 1.7833593749999996
Лагранж:
-62.5001*x^4 + 55.8335*x^3 + -3.8751*x^2 + 9.4917*x^1 + 0.29
Ньютон:
-62.5*x^4 + 55.8333*x^3 + -3.875*x^2 + 9.4917*x^1 + 0.29
```

Выводы

В результате выполнения данной лабораторной работы были изучены методы для интерполяции функций, с помощью которой можно находить значения функции в областях, где она не определена.