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 Численное дифференцирование

Студент Кононенко С.С.

Группа ИУ7-43Б

Оценка (баллы) ________________

Преподаватель Градов В.М.

Москва — 2020 г.

1. Задание

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

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

параметры функции неизвестны **и определять их не нужно**.

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					

Вычислить первые разностные производные от функции и занести их в столбцы (1)-(4) таблицы:

1 - односторонняя разностная производная,

2 - центральная разностная производная,

3 - 2-я формула Рунге с использованием односторонней производной,

4 - введены выравнивающие переменные.

В столбец 5 занести вторую разностную производную.

Вход: приведенная выше таблица.
Выход: заполненная таблица.

2. Описание алгоритма

Используя разложение в ряд Тейлора можно получить левую

$$y'_n = \frac{y_{n+1} - y_n}{h}$$

и правую разностную формулы

$$y'_n = \frac{y_n - y_{n-1}}{h}$$

Данные формулы имеют самый низкий - первый - порядок точности. Из данных формул можно получить центральную разностную формулу

$$y'_n = \frac{y_{n+1} - y_{n-1}}{h}$$

Центральная разностная формула имеет второй порядок точности.

Приведенные выше формулы имеют погрешность вида $R = \psi(x)h^p$. С помощью преобразований в рядах Тейлора можно получить первую формулу Рунге

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

Отсюда можно получить вторую формулу Рунге

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

Формулы Рунге справедливы не только для операций дифференцирования, но и для других приближенных вычислений (при условии, что погрешность формул имеет вышеприведенный вид).

Помимо приведенных выше методов стоит отметить метод, заключающийся в применении выравнивающих переменных. При правильном подборе исходная кривая может быть преобразована в прямую, производная от которой вычисляется точно даже по простым формулам.

Пусть задана функция $y(x)$ с введенными переменными $\xi = \xi(x)$ и $\eta = \eta(y)$. Тогда возврат к заданным переменным будет осуществлен по формуле

$$y'_x = \frac{\eta'_\xi \xi'_x}{\eta'_y}$$

3. Результаты работы программы

x	y	1	2	3	4	5
1	0.571	nil	nil	nil	0.408	nil
2	0.889	0.318	0.260	nil	0.247	-0.116
3	1.091	0.202	0.171	0.144	0.165	-0.062
4	1.231	0.140	0.121	0.109	0.118	-0.038
5	1.333	0.102	0.090	0.083	0.089	-0.023
6	1.412	0.079	nil	0.067	nil	nil

Первый столбец - левосторонняя формула (точность $O(h)$).
Второй столбец - центральная формула (точность $O(h^2)$).
Третий столбец - вторая формула Рунге (с использованием левосторонней формулы).
Четвертый столбец - применение выравнивающих переменных (оценка точность сложна, так как неизвестны параметры). Использовано соотношение

$$y'_x = \frac{\eta'_\xi y^2}{x^2}$$

Пятый столбец - вторая разностная производная.

4. Ответы на вопросы для защиты лабораторной работы

1. Получить формулу порядка точности $O(h^2)$ для первой разностной производной y_N в крайнем правом узле x_N.

$$y_{N-1} = y_N - hy'_n + \frac{h^2}{2!}y''_N - \frac{h^3}{3!}y'''_N ...$$

$$y_{N-2} = y_N - 2hy'_n + \frac{4h^2}{2!}y''_N - \frac{8h^3}{3!}y'''_N ...$$

Отсюда

$$y'_N = \frac{3y_N - 4y_{N-1} + y_{N-2}}{2h} + \frac{h^2}{3}y'''_N$$

Полученная формула

$$y'_N = \frac{3y_N - 4y_{N-1} + y_{N-2}}{2h} + O(h^2)$$

2. Получить формулу порядка точности $O(h^2)$ для второй разностной производной y''_0 в крайнем левом узле x_0.

$$y_1 = y_0 + hy'_0 + \frac{h^2}{2!}y''_0 - \frac{h^3}{3!}y'''_0 ...$$

$$y_2 = y_0 + 2hy'_0 + \frac{4h^2}{2!}y''_0 - \frac{8h^3}{3!}y'''_0 ...$$

Произведем преобразования - домножим y_1 на 4 и вычтем y_2

$$4y_1 - y_2 = 4y_0 - y_0 + 2hy'_0 + O(h^2)$$

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

Полученная формула

$$y''_0 = \frac{-y_3 + 4y_2 - 5y_1 + 2y_0}{h^2} + O(h^2)$$

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

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

$$= \frac{y_{n+1} - y_n}{h} + \frac{\frac{y_{n+1}-y_n}{h} - \frac{y_{n+2}-y_n}{2h}}{2^1 - 1} + O(h^2) =$$

$$= \frac{-3y_n + 4y_{n+1} - y_{n+2}}{2h} + O(h^2)$$

Для левого узла – $n = 0, n + 1 = 1, n + 2 = 2 \Rightarrow y_0' = \frac{-3y_0 + 4y_1 - y_2}{2h} + O(h^2)$.

4. Любым способом из Лекций №7, 8 получить формулу порядка точности $O(h^3)$ для первой разностной производной y_0' в крайнем левом узле x_0.

$$y_1 = y_0 + hy_0' + \frac{h^2}{2!}y_0'' + \frac{h^3}{3!}y_0'''...$$
$$y_2 = y_0 + 2hy_0' + \frac{4h^2}{2!}y_0'' + \frac{8h^3}{3!}y_0'''...$$
$$y_3 = y_0 + 3hy_0' + \frac{9h^2}{2!}y_0'' + \frac{27h^3}{3!}y_0'''...$$

Произведем преобразования

$$y' = \frac{y_3 + 27y_1 - 28y_0}{30h} + O(h^3)$$

5. Код программы

5.1. Основной пакет приложения.

```
package main

import "./differentiate"

func main() {
  x := []float64{1, 2, 3, 4, 5, 6}
  y := []float64{0.571, 0.889, 1.091, 1.231, 1.333, 1.412}
  h := 1.0

  differentiate.FmtPrintInit("X              :", x)
  differentiate.FmtPrintInit("Y              :", y)
  differentiate.FmtPrintRes("Onesided       :", differentiate.LeftDiff(y, h))
  differentiate.FmtPrintRes("Center         :", differentiate.CenterDiff(y, h))
  differentiate.FmtPrintRes("Second Runge   :", differentiate.SecondRungeDiff(y, h, 1))
  differentiate.FmtPrintRes("Aligned params :", differentiate.AlignedCoeffsDiff(x, y))
  differentiate.FmtPrintRes("Second onesided:", differentiate.SecondLeftDiff(y, h))
}
```

5.2. Пакет реализации численного дифференцирования.

5.2.1. Хранение данных.

```
package differentiate

// DFloat64 used to represent value and derivative existance.
type DFloat64 struct {
  value float64
  null  bool
}

func (n *DFloat64) retvalue() interface{} {
  if n.null {
    return nil
  }
  return n.value
}

func newVal(x float64) DFloat64 {
  return DFloat64{
    value: x,
    null:  false,
  }
}

func newNil() DFloat64 {
```

```
  return DFloat64{
    value: 0.,
    null:  true,
  }
}
```

5.2.2. Реализация численного дифференцирования.

```
package differentiate

import (
  "fmt"
  "math"
)

func leftDiffInter(y, yl, h float64) float64 {
  return (y - yl) / h
}

// LeftDiff used to represent onesided subtractive derivative.
func LeftDiff(y []float64, h float64) (res []DFloat64) {
  for i := range y {
    if i == 0 {
      res = append(res, newNil())
    } else {
      res = append(res, newVal(leftDiffInter(y[i], y[i-1], h)))
    }
  }
  return
}

// CenterDiff used to represent center derivative.
func CenterDiff(y []float64, h float64) (res []DFloat64) {
  for i := range y {
    if i == 0 || i == len(y)-1 {
      res = append(res, newNil())
    } else {
      res = append(res, newVal((y[i+1]-y[i-1])/2*h))
    }
  }
  return
}

// SecondRungeDiff used to represent second Runge derivative.
func SecondRungeDiff(y []float64, h, p float64) (res []DFloat64) {
  var y2h []DFloat64

  for i := range y {
    if i < 2 {
```

```go
      y2h = append(y2h, newVal(0))
    } else {
      y2h = append(y2h, newVal((y[i]-y[i-2])/(2.*h)))
    }
  }

  yh := LeftDiff(y, h)
  for i := range yh {
    if i < 2 {
      res = append(res, newNil())
    } else {
      res = append(res, newVal(yh[i].value+(yh[i].value-y2h[i].value)/
        (math.Pow(2., p)-1)))
    }
  }
  return
}

// AlignedCoeffsDiff used to represent derivative with aligned parameters.
func AlignedCoeffsDiff(x, y []float64) (res []DFloat64) {
  for i := range y {
    if i == len(y)-1 {
      res = append(res, newNil())
    } else {
      k := y[i] * y[i] / x[i] / x[i]
      res = append(res,
        newVal(k*leftDiffInter(-1./y[i+1], -1./y[i], -1./x[i+1]- -1./x[i])))
    }
  }
  return
}

// SecondLeftDiff used to represent second subtractive derivative.
func SecondLeftDiff(y []float64, h float64) (res []DFloat64) {
  for i := range y {
    if i == 0 || i == len(y)-1 {
      res = append(res, newNil())
    } else {
      res = append(res, newVal((y[i-1]-2*y[i]+y[i+1])/(h*h)))
    }
  }
  return
}

// FmtPrintInit used to init formatted output.
func FmtPrintInit(text string, init []float64) {
  fmt.Printf("%s ", text)
  for _, v := range init {
```

```
    fmt.Printf("%7.4f ", v)
  }
  fmt.Printf("\n")
}

// FmtPrintRes used to init formatted output.
func FmtPrintRes(text string, res []DFloat64) {
  fmt.Printf("%s ", text)
  for _, v := range res {
    fmt.Printf("%7.4f ", v.retvalue())
  }
  fmt.Printf("\n")
}
```