Университет ИТМО
Факультет программной инженерии и компьютерной техники

# Вычислительная математика. Лабораторная работа №3. Численное интегрирование

Группа: P32131
Студент: Смирнов Виктор Игоревич
Вариант: 17

2023

## Ключевые слова

Численные методы, интегрирование.

# 1 Цель работы

Цель данной работы - найти приближенное значение определенного интеграла с требуемой точностью различными численными методами, реализовав численные методы на языке программирования.

# 2 Вычислительная часть

Необходимо вычислить интеграл функции $y = 3x^3 - 4x^2 + 7x - 17$ на интервале [1, 2].

## 2.1 Вычисление вручную

$$f(x) = 3x^3 - 4x^2 + 7x - 17 \tag{1}$$

$$\int f(x) = 3x^3 - 4x^2 + 7x - 17 = \frac{3x^4}{4} - \frac{4x^3}{3} + \frac{7x^2}{2} - 17x + const \tag{2}$$

$$F(x) = \frac{3x^4}{4} - \frac{4x^3}{3} + \frac{7x^2}{2} - 17x \tag{3}$$

$$\int_1^2 f(x) = F(2) - F(1) = -\frac{55}{12} = -4.583(3) \tag{4}$$

## 2.2 Вычисление по формуле Ньютона-Котеса при n = 5

```
$ math-tool integrator cotes5 1 2 0.0001
6
Report
Taken method: Cotes n = 5
Scope: [1.000000, 2.000000]
Accuracy: 0.0001
Result: -4.58341
```

Листинг 1: Результаты вывода программы 1

Таблица 1: Трассировка метода Котеса (n = 5)

| n | size | prev | curr | diff |
|---|---|---|---|---|
| 1 | 8 | -4.92739 | -4.75271 | 0.174676 |
| 2 | 16 | -4.75271 | -4.66735 | 0.0853653 |
| 3 | 32 | -4.66735 | -4.62517 | 0.0421784 |
| 4 | 64 | -4.62517 | -4.60421 | 0.0209617 |
| 5 | 128 | -4.60421 | -4.59376 | 0.0104488 |
| 6 | 256 | -4.59376 | -4.58854 | 0.00521638 |
| 7 | 512 | -4.58854 | -4.58594 | 0.00260618 |
| 8 | 1024 | -4.58594 | -4.58464 | 0.00130259 |
| 9 | 2048 | -4.58464 | -4.58398 | 0.000651167 |
| 10 | 4096 | -4.58398 | -4.58366 | 0.000325552 |
| 11 | 8192 | -4.58366 | -4.5835 | 0.000162768 |
| 12 | 16384 | -4.5835 | -4.58341 | 8.13822e-05 |

## 2.3 Вычисление методом средних прямоугольников

```
$ math-tool integrator rect-m 1 2 0.0001
6
Report
Taken method: Rectangle Middle
Scope: [1.000000, 2.000000]
Accuracy: 0.0001
Result: -4.58335
```

Листинг 2: Результаты вывода программы 2

Таблица 2: Трассировка метода Средних Прямоугольников

| n | size | prev | curr | diff |
|---|---|---|---|---|
| 1 | 8 | -4.63281 | -4.5957 | 0.0371094 |
| 2 | 16 | -4.5957 | -4.58643 | 0.00927734 |
| 3 | 32 | -4.58643 | -4.58411 | 0.00231934 |
| 4 | 64 | -4.58411 | -4.58353 | 0.000579834 |
| 5 | 128 | -4.58353 | -4.58338 | 0.000144958 |
| 6 | 256 | -4.58338 | -4.58335 | 3.62396e-05 |

## 2.4 Вычисление по формуле Трапеций

```
$ math-tool integrator trapeze 1 2 0.0001
6
Report
Taken method: Trapeze
Scope: [1.000000, 2.000000]
Accuracy: 0.0001
Result: -4.58331
```

Листинг 3: Результаты вывода программы 3

Таблица 3: Трассировка метода Трапеций

| n | size | prev | curr | diff |
|---|---|---|---|---|
| 1 | 8 | -4.48438 | -4.55859 | 0.0742188 |
| 2 | 16 | -4.55859 | -4.57715 | 0.0185547 |
| 3 | 32 | -4.57715 | -4.58179 | 0.00463867 |
| 4 | 64 | -4.58179 | -4.58295 | 0.00115967 |
| 5 | 128 | -4.58295 | -4.58324 | 0.000289917 |
| 6 | 256 | -4.58324 | -4.58331 | 7.24792e-05 |

## 2.5 Вычисление по формуле Симпсона

```
$ math-tool integrator simpson 1 2 0.0001
6
Report
Taken method: Simpson
Scope: [1.000000, 2.000000]
Accuracy: 0.0001
Result: -4.58333
```

Листинг 4: Результаты вывода программы 4

## 2.6 Функция с разрывом - расходящийся интеграл

Таблица 4: Трассировка метода Симпсона

| n | size | prev | curr | diff |
|---|---|---|---|---|
| 1 | 8 | -4.58333 | -4.58333 | 1.77636e-15 |

```
$ math-tool integrator simpson -1 2 0.0001
7
Report
Taken method: Simpson
Scope: [-1.000000, 2.000000]
Accuracy: 0.0001
No result: Can't integrate given function, maybe it diverges in given interval
```

Листинг 5: Результаты вывода программы при расходящемся интеграле 1

# 3 Анализ полученных результатов

Начнем со случая, когда функция терпит разрыв в точке на интервале. Если несобственный интеграл расходится, то его интеграл стремится к бесконечности, поэтому при каждом новом приближении мы получаем все большее и большее число в результате, поэтому я просто ограничил количество интераций 20ю, и если это кол-во превышено кидаю ошибку.

В целом, сразу видно, насколько хорош Симпсон, хотя это на самом деле из-за того, что наша исходная функция является полиномом 4ой степени и данный метод своими приближениями очень точно подходит под данную модель.

Метод трапеции и прямоугольников работают примерно одинаково. А вот формула Ньютона-Котеса при n = 5 расстроила - не стоила она потраченных сил. А все почему? Потому что интерполяционные многочлены подвели.

# 4 Фрагменты программ

```
template <typename T>
using TrivialMethod = std::function<T(Function<T>, Interval<T>)>;

template <typename T>
using Method = std::function<T(Function<T>, Partition<T>)>;

template <typename T>
using PointPeeker = std::function<T(Interval<T>)>;

template <typename T>
TrivialMethod<T> trivial0(PointPeeker<T> peek) noexcept {
  return [peek](auto f, auto interval) {
    return f(peek(interval)) * interval.length();
  };
}

template <typename T>
TrivialMethod<T> trivial1() noexcept {
  return [](auto f, auto interval) {
    auto left = f(interval.left());
    auto right = f(interval.right());
    return (left + right) / 2 * interval.length();
  };
}

template <typename T>
TrivialMethod<T> trivial2() noexcept {
  return [](auto f, auto interval) {
    auto a = interval.left();
    auto b = interval.right();
    return (b - a) / 6 * (
        + 1 * f(a)
        + 4 * f((a + b) / 2)
        + 1 * f(b)
    );
```

```
  };
}

template <typename T>
TrivialMethod<T> trivial5() noexcept {
  return [](auto f, auto interval) {
    const auto s = interval.left();
    const auto l = interval.length() / 6;
    return interval.length() / 288 * (
        + 19 * f(s + 0 * l)
        + 75 * f(s + 1 * l)
        + 50 * f(s + 2 * l)
        + 50 * f(s + 3 * l)
        + 75 * f(s + 4 * l)
        + 19 * f(s + 5 * l)
    );
  };
}

template <typename T>
class Cotes {
public:
  explicit Cotes(TrivialMethod<T> trivial)
      : trivialAreaUnderGraph(trivial) {}

  T areaUnderGraph(Function<T> function, Partition<T> partition) noexcept {
    T sum = 0;
    for (auto interval : partition) {
      sum += trivialAreaUnderGraph(function, interval);
    }
    return sum;
  }

private:
  TrivialMethod<T> trivialAreaUnderGraph;
};

template <typename T>
class Approx {
public:
  explicit Approx(T epsilon, Cotes<T> cotes)
      : epsilon(epsilon), cotes(cotes) {}

  T integrate(Function<T> f, Interval<T> scope) {
    auto size = 4;

    auto prev = cotes.areaUnderGraph(f, Partition<T>(scope, size));
    for (Index i = 0; i < iterationsLimit; i++) {
      size *= 2;

      auto curr = cotes.areaUnderGraph(f, Partition<T>(scope, size));

      if (std::abs(prev - curr) < epsilon) {
        return curr;
      }

      prev = curr;
    };
    throw Error::ProcessDiverges(
        "Can't integrate given function, "
        "maybe it diverges in given interval"
    );
  }

private:
  Count iterationsLimit = 20;
  T epsilon;
  Cotes<T> cotes;
};
```

Листинг 6: Программы для численного интегрирования

# 5 Вывод

Выполняя данную лабораторную работу я познакомился с основными численными методами интегрирования.

Интегрирование - вычислительно сложная задача, то есть для нее требуется много ресурсов. Для сильно скачущих функциях требуются довольно мелкое разбиение из-за чего колчество итераций сильно возрастает. Кроме того мы увеличиваем количество этих итераций при каждой неудаче.

С расходящимися интегралами могут быть проблемы: мы можем попытаться вычислить значение функции на интервале, на котором она не определена и получить Undefined Behaviour, а если она будет просто стремится к бесконечности на заданном интервале, мы просто потратим много сил на вычисление и в конце-концов сдадимся.