ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

tpu.ru

Углубленный курс информатики

Лекция 10

Численные методы интегрирования и решения обыкновенных дифференциальных уравнений

29 апреля 2022 г.

Вячеслав Алексеевич Чузлов,
к.т.н., доцент ОХИ ИШПР

ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Содержание

ТОМСКИЙ
ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

Численные методы интегрирования

Численные методы интегрирования

- Большое число научно-технологических задач требует объединения в математическое описание всей информации о процессе.
- Например, для математических моделей химико-технологических процессов одними из основных характеризующих параметров являются концентрации реагентов, температура и др.
- В большинстве случаев балансовые уравнения в химической технологии представлены системой интегральных и дифференциальных уравнений.
- К примеру, интегральные уравнения можно встретить при описании гетерогенной кинетики, в теории активированного комплекса, при моделировании работы химических реакторов и т.д.
- Во многих случаях на практике бывает сложно вычислить интеграл аналитически, поэтому применяются методы приближенного численного интегрирования.

Постановка задачи

Пусть требуется вычислить определенный интеграл:

$$I = \int_a^b f(x)\,dx \tag{1}$$

при условии, что a и b конечны и $f(x)$ – непрерывная функция x на всем интервале $x \in [a, b]$.
Если подынтегральная функция задана в аналитическом виде, то в большинстве случаев интеграл от этой функции в пределах от a до b можно вычислить, используя формулу *Ньютона-Лейбница*:

$$\int_a^b f(x)\,dx = F(x)\Big|_a^b = F(b) - F(a) \tag{2}$$

К сожалению, формулой (2) во многих случаях нельзя воспользоваться по следующим причинам:

- первообразная функции $f(x)$ слишком сложная или ее вовсе нельзя выразить аналитически через элементарные функции;
- функция $f(x)$ задана в табличном виде, что особенно актуально для задач химической технологии, в особенности, при обработке экспериментальных данных.

В подобных случаях используют методы численного интегрирования, задача которых состоит в нахождении приближенного значения интеграла (1) по заданным или вычисленным значениям.

Постановка задачи

Общий подход к решению задачи численного интегрирования можно описать следующим образом.

- Определенный интеграл – это площадь фигуры под графиком функции $f(x)$, ограниченная осью OX и переменными $x = a$ и $x = b$.
- Для приближенного вычисления интеграла необходимо разбить интервал $[a, b]$ на множество более мелких интервалов, определяя площадь каждой получившейся фигуры и суммируя их.
- В зависимости от способа вычисления подынтегральной суммы существуют различные методы численного интегрирования (методы прямоугольников, трапеций, парабол и т.д.).

Метод прямоугольников

Наиболее простым методом численного интегрирования является *метод прямоугольников*. Суть данного метода заключается в замене определенного интеграла интегральной суммой:

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} f\left(x_i^*\right) \cdot \Delta x_i; \qquad x_i^* \in [x_{i-1}, x_i] \tag{3}$$

Разобьем интервал интегрирования $[a, b]$ на n равных частей. Введем обозначение $\Delta x_i = h$ – шаг разбиения. Формула прямоугольника применяется к каждой части интервала интегрирования. В качестве точек x_i^* выбирают левые ($x_i^* = x_{i-1}$) или правые ($x_i^* = x_i$) границы элементарных отрезков (рисунки 1, 2).
Таким образом, формулы для левых и правых прямоугольников можно записать следующим образом:

$$\int_a^b f(x)\,dx = h_1 \cdot f\left(x_0\right) + h_2 \cdot f\left(x_1\right) + \ldots + h_n \cdot f\left(x_{n-1}\right) \tag{4}$$

$$\int_a^b f(x)\,dx = h_1 \cdot f\left(x_1\right) + h_2 \cdot f\left(x_2\right) + \ldots + h_n \cdot f\left(x_n\right) \tag{5}$$

Метод прямоугольников

Наиболее точным является метод средних прямоугольников, использующий значения функции в средних точках элементарных отрезков $\overline{x}_i$ (рисунок 3). Так, площадь криволинейной трапеции заменяется суммой площадей прямоугольников с основанием h и высотами, равными значениям функции $f(x)$ в середине оснований этих прямоугольников $f(\overline{x}_i)$.
В итоге получим формулу:

$$\int_a^b f(x)\,dx = \frac{b-a}{n} \cdot \sum_{i=1}^{n} f(\overline{x}_i) \tag{6}$$

где $\frac{b-a}{n} = h$ или

$$\int_a^b f(x)\,dx \approx h \cdot \sum_{i=1}^{n} f\left(\frac{x_{i+1}+x_i}{2}\right) \tag{7}$$

Метод прямоугольников

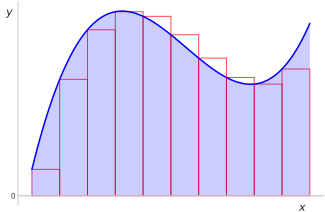

Рисунок 1 – Метод левых прямоугольников

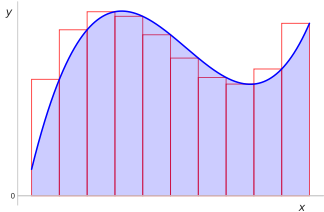

Рисунок 2 – Метод правых прямоугольников

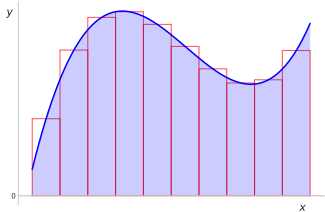

Рисунок 3 – Метод средних прямоугольников

Пример

Вычислим интеграл методом прямоугольников:

$$I = \int_0^1 \frac{dx}{1+x^2}$$

Интервал интегрирования разобьем на 10 равных частей, т.е. примем $n = 10$.
Первым делом определим h:

$$h = \frac{b-a}{n} = \frac{1-0}{10} = 0.1$$

Далее для более удобной программной реализации перепишем формулу (7) в следующем виде:

$$\int_a^b f(x)\,dx = h \cdot \sum_{i=0}^{n-1} f\left(a + h \cdot (i + 0.5)\right) \tag{8}$$

где выражение $(a + h \cdot (i + 0.5))$ используется для определения середин прямоугольников.

Пример

По формуле (8) определим искомую сумму:

$$\int_0^1 \frac{dx}{1+x^2} = 0.1 \cdot \sum_{i=0}^{10-1} \frac{1}{1+(0+0.1\cdot(i+0.5))^2} =$$

$$= 0.1 \cdot \left(\frac{1}{1+(0.1\cdot(0+0.5))^2} + \frac{1}{1+(0.1\cdot(1+0.5))^2} + \frac{1}{1+(0.1\cdot(2+0.5))^2} + \right.$$
$$+ \frac{1}{1+(0.1\cdot(3+0.5))^2} + \frac{1}{1+(0.1\cdot(4+0.5))^2} + \frac{1}{1+(0.1\cdot(5+0.5))^2} +$$
$$+ \frac{1}{1+(0.1\cdot(6+0.5))^2} + \frac{1}{1+(0.1\cdot(7+0.5))^2} + \frac{1}{1+(0.1\cdot(8+0.5))^2} +$$
$$\left. + \frac{1}{1+(0.1\cdot(9+0.5))^2} \right) = 0.7856$$

Таким образом, приближенное значение искомого интеграла по методу средних прямоугольников равно 0.7856.

Программная реализация


```python
def func(x):
    return 1 / (1 + x ** 2)


def rects(func, limits, n=10):
    a, b = limits

    #Определение шага разбиения
    h = (b - a) / n

    #Расчет подынтегральной суммы
    s = 0
    for i in range(n):
        s += h * func(a + h * (i + 0.5))

    return s


print(rects(func, [0, 1])) #0.7856064962502745

```

Программная реализация

Альтернативный вариант реализации функции `rects` не содержит цикл `for`, вместо этого используется генераторное выражение, что немного повышает производительность кода при высоких значениях `n`:

```
1  def rects(func, limits, n=10):
2      a, b = limits
3      h = (b - a) / n
4
5      return sum(func(a + h * (i + 0.5)) for i in range(n)) * h
6
```

Альтернативный вариант функции `rects` подразумевает также вызов встроенной функции `sum` взамен объявления вспомогательной переменной `s` для хранения значения подынтегральной суммы.

Метод трапеций

Метод трапеций основан на использовании линейной интерполяции. Иначе говоря, график функции $y = f(x)$ заменяется ломаной линией, соединяющей точки (x_i, y_i). При таком подходе общая площадь криволинейной трапеции будет являться суммой всех площадей элементарных прямоугольных трапеций (рисунок 4).

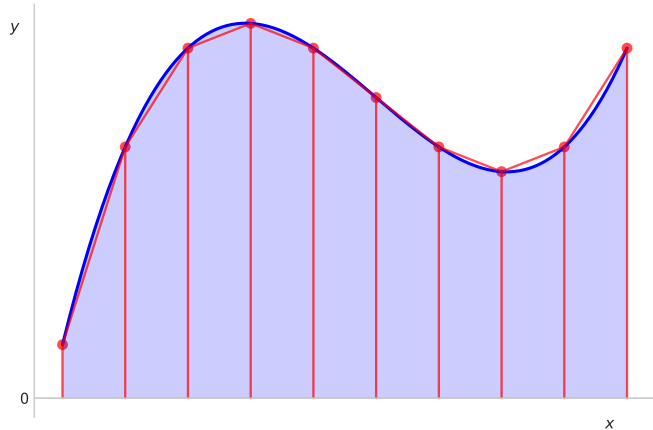

Рисунок 4 – Метод трапеций

Метод трапеций

Площадь каждой элементарной прямоугольной трапеции можно определить по формуле:

$$S_i = \frac{y_{i-1} + y_i}{2} \cdot h, \qquad i = 1, 2, \ldots, n \tag{9}$$

где $h = \dfrac{b-a}{n}$, n – число интервалов разбиения.

Таким образом, просуммировав все площади элементарных трапеций, получим формулу для *метода трапеций*:

$$\int_a^b f(x)\,dx = \sum_{i=1}^{n} S_i = \frac{h}{2} \cdot \sum_{i=1}^{n} (y_{i-1} + y_i) \tag{10}$$

или

$$\int_a^b f(x)\,dx = h \cdot \sum_{i=1}^{n} \frac{f(x_{i-1}) + f(x_i)}{2} \tag{11}$$

Пример

Рассмотрим решение примера, приведенного на слайде 11, с использованием метода трапеций. Сначала определим величину h:

$$h = \frac{b-a}{n} = \frac{1-0}{10} = 0.1$$

Затем необходимо найти массив значений x на заданном интервале интегрирования $[a, b]$ с шагом h:

$$x_i = a + i \cdot h; \quad i = 0, 1, 2, \ldots, n, n+1$$

Далее воспользуемся формулой (11) и найдем приближенное значение интеграла:

$$\int_0^1 \frac{dx}{1+x^2} = 0.1 \cdot \sum_{i=0}^{n} \frac{\frac{1}{1+x_i^2} + \frac{1}{1+x_{i+1}^2}}{2} = 0.1 \cdot \left(\frac{\frac{1}{1+0^2} + \frac{1}{1+0.1^2}}{2} + \frac{\frac{1}{1+0.1^2} + \frac{1}{1+0.2^2}}{2} + \right.$$

$$+ \frac{\frac{1}{1+0.2^2} + \frac{1}{1+0.3^2}}{2} + \frac{\frac{1}{1+0.3^2} + \frac{1}{1+0.4^2}}{2} + \frac{\frac{1}{1+0.4^2} + \frac{1}{1+0.5^2}}{2} + \frac{\frac{1}{1+0.5^2} + \frac{1}{1+0.6^2}}{2} +$$

$$\left. \frac{\frac{1}{1+0.6^2} + \frac{1}{1+0.7^2}}{2} + \frac{\frac{1}{1+0.7^2} + \frac{1}{1+0.8^2}}{2} + \frac{\frac{1}{1+0.8^2} + \frac{1}{1+0.9^2}}{2} + \frac{\frac{1}{1+0.9^2} + \frac{1}{1+1^2}}{2} \right) = 0.78498$$

Программная реализация


```python
def func(x):
    return 1 / (1 + x ** 2)


def traps(func, limits, n=10):
    a, b = limits
    h = (b - a) / n

    #Заполним значения x
    x = [a + i * h for i in range(n+1)]

    #Расчет подынтегральной суммы
    s = 0
    for i in range(n):
        s += h * (func(x[i]) + func(x[i+1])) / 2

    return s


print(traps(func, [0, 1])) #0.7849814972267897

```

Программная реализация

Так же, как и в случае с методом прямоугольников, цикл `for` в функции `traps` можно заменить генераторным выражением, что даст небольшой прирост производительности при высоких значениях числа интервалов разбиения `n`:

```
1  def traps(func, limits, n=10):
2      a, b = limits
3      h = (b - a) / n
4
5      #Заполним значения x
6      x = [a + i * h for i in range(n+1)]
7
8      return sum((func(x[i]) + func(x[i+1])) / 2 for i in range(n)) * h
9
```

Альтернативный вариант функции `traps` подразумевает также вызов встроенной функции `sum` взамен объявления вспомогательной переменной `s` для хранения значения подынтегральной суммы.

Метод парабол (формула Симпсона)

Данный метод является наиболее точным в сравнении с методами прямоугольников и трапеций. Формула Симпсона основана на квадратичной интерполяции подынтегральной функции на отрезке $[a, b]$ по трем равноотстоящим узлам.
Разобьем интервал интегрирования $[a, b]$ на n равных отрезков величины h. Примем:

$$x_0 = a, \quad x_1 = x_0 + h, \quad \ldots, \quad x_n = x_0 + n \cdot h = b.$$

Тогда значения функции в этих точках обозначим как:

$$y_0 = f(a), \quad y_1 = f(x_1), \quad y_2 = f(x_2), \quad \ldots y_n = f(b)$$

Далее заменим подынтегральную функцию $f(x)$ интерполяционным многочленом второй степени на каждом отрезке $[x_0, x_2], [x_2, x_4], \ldots, [x_{i-1}, x_{i+1}]$ (рисунок 5):

$$f(x) \approx P_i(x) = a_i \cdot x^2 + b_i \cdot x + c_i \tag{12}$$

где $x_{i-1} \leqslant x \leqslant x_{i+1}$.

Метод парабол (формула Симпсона)

В качестве интерполяционного полинома второй степени $P_i(x)$ может быть использован полином Лагранжа, проходящий через концы каждых из трех ординат:

$$y_0, y_1, y_2; \quad y_2, y_3, y_4;$$

$$y_4, y_5, y_6; \quad \dots; \quad y_{n-2}, y_{n-1}, y_n$$

Полином Лагранжа для интервала $[x_{i-1}, x_{i+1}]$ записывается следующим образом:

$$P_i(x) = \frac{(x - x_i) \cdot (x - x_{i+1})}{(x_{i-1} - x_i) \cdot (x_{i-1} - x_{i+1})} \cdot y_{i-1} + \\ + \frac{(x - x_{i-1}) \cdot (x - x_{i+1})}{(x_i - x_{i-1}) \cdot (x_i - x_{i+1})} \cdot y_i + \\ + \frac{(x - x_{i-1}) \cdot (x - x_i)}{(x_{i+1} - x_{i-1}) \cdot (x_{i+1} - x_i)} \cdot y_{i+1} \quad (13)$$

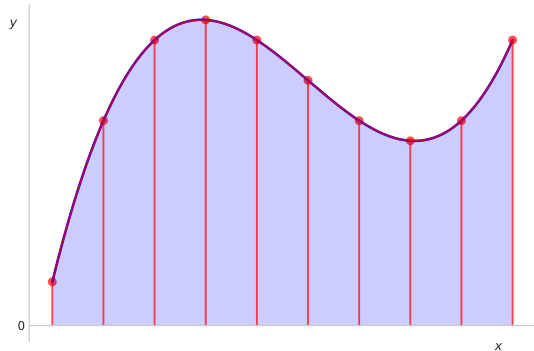

Рисунок 5 – Метод парабол (формула Симпсона)

Метод парабол (формула Симпсона)

Взяв интеграл от интерполяционного полинома второй степени на отрезке $[a, b]$, получим формулу Симпсона:

$$\int_a^b f(x)\,dx \approx \int_a^b P_2(x)\,dx = \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right) \tag{14}$$

где $f(a)$, $f((a+b)/2)$ и $f(b)$ – значения функции в соответствующих точках (на концах отрезка и в его середине).

В целях более точного вычисления интеграла, отрезок $[a, b]$ разбивается на $N = 2n$ элементарных отрезков равной длины и применяется формула Симпсона на составных отрезках. Каждый из составных отрезков состоит из соседней пары элементарных отрезков. Результирующее значение является суммой результатов интегрирования на составных отрезках:

$$\int_a^b f(x)\,dx \approx \frac{h}{3}\left[f(x_0) + 2\sum_{j=1}^{N/2-1} f(x_{2j-1}) + 4\sum_{j=1}^{N/2} f(x_{2j}) + f(x_N)\right] \tag{15}$$

где $h = \dfrac{b-a}{N}$ – величина шага; $x_j = a + jh$ – чередующиеся границы и середина отрезков, на которых применяется формула Симпсона. Каждый составной отрезок $\left[x_{j-1}, x_{j+1}\right]$ состоит из элементарных отрезков $\left[x_{j-1}, x_j\right]$, $\left[x_j, x_{j+1}\right]$.

Пример

Рассмотрим решение примера, приведенного на слайде 11, только в этот раз воспользуемся формулой Симпсона.

Сначала определим величину h:

$$h = \frac{b-a}{n} = \frac{1-0}{10} = 0.1$$

Результаты вычисления подынтегральной функции представлены в таблице:

x_i	$f(x_i) \quad i = 1, 3, \ldots$	$f(x_i) \quad i = 2, 4, \ldots$	$f(x_0), f(x_{10})$
0.0			1.0
0.1	0.99010		
0.2		0.96154	
0.3	0.91743		
0.4		0.86207	
0.5	0.80000		
0.6		0.73529	
0.7	0.67114		
0.8		0.60976	
0.9	0.55249		
1.0			0.5

Пример

Воспользуемся формулой (15):

$$\int_0^1 \frac{dx}{1+x^2} = \frac{0.1}{3}\big(1.0 + 2\cdot(0.99010 + 0.91743 + 0.80000 + 0.67114 + 0.55249) + \\ + 4\cdot(0.96154 + 0.86207 + 0.73529 + 0.60976) + 0.5\big) = 0.78540$$

Таким образом, искомое значение определенного интеграла по методу Симпсона равно 0.78540.

Программная реализация


```python
def func(x):
    return 1 / (1 + x ** 2)


def simpson(func, limits, n=10):
    a, b = limits
    h = (b - a) / n

    x = [a + i * h for i in range(1, n)]  #x в диапазоне (a, b) с шагом h

    #Расчет подынтегральной суммы
    #s1 - сумма элементов с нечетными индексами
    #s2 - сумма элементов с четными индексами
    s1, s2 = 0, 0
    for i in range(len(x)):
        if i % 2:
            s1 += func(x[i])
        else:
            s2 += func(x[i])

    return h / 3 * (func(a) + 2 * s1 + 4 * s2 + func(b))


print(simpson(func, [0, 1])) #0.7853981534848038

```

Программная реализация

- Условный оператор в приведенном выше примере можно заменить операцией взятия среза, т.к. в данном случае оператор `if` всего лишь разделяет элементы массива `x` с нечетными и четными индексами.
- Если нужны элементы с четными индексами, можно использовать следующую форму операции среза `x[::2]`, т.е. взять все элементы массива `x` с первого до последнего, но с шагом 2.
- Напротив, если нужны элементы с нечетными индексами, то можно использовать следующую форму операции взятия среза: `x[1::2]`:

```
1  def simpson(func, limits, n=10):
2      a, b = limits
3      h = (b - a) / n
4
5      x = [a + i * h for i in range(1, n)]  #x в диапазоне (a, b) с шагом h
6
7      #Расчет подынтегральной суммы
8      #s1 - сумма элементов с четными индексами
9      #s2 - сумма элементов с нечетными индексами
10     s1 = sum(func(element) for element in x[1::2])
11     s2 = sum(func(element) for element in x[::2])
12
13     return h / 3 * (func(a) + 2 * s1 + 4 * s2 + func(b))
14
```

ТОМСКИЙ
ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

Решение обыкновенных дифференциальных уравнений

Решение обыкновенных дифференциальных уравнений

- Обыкновенные дифференциальные уравнения широко распространенны в области математического моделирования химико-технологических процессов. Например, с их помощью моделируется кинетика химических реакций, процессы, протекающие в химических реакторах, в массо- и теплообменных аппаратах.
- В дифференциальных уравнениях устанавливается связь между независимыми переменными, искомыми функциями и их производными. В тех случаях, когда искомая функция зависит от одной переменной, дифференциальные уравнения называются *обыкновенными*.

В качестве примера можно привести описание структуры движения потока в реакторе идеального смешения, представленное обыкновенным дифференциальным уравнением:

$$\frac{dC}{dt} = \frac{1}{\tau}(C_0 - C)$$

При этом для реактора идеального вытеснения структура потока описывается уравнением в частных производных:

$$\frac{\partial C}{\partial t} = -U\frac{\partial C}{\partial l}$$

при этом функция $C(t, l)$ зависит от времени (t) и от длины аппарата (l).

Обыкновенные дифференциальные уравнения (ОДУ) – это уравнения, содержащие одну или несколько производных от искомой функции $y = f(x)$:

$$F\left(x, y, y', \ldots, y^{(n)}\right) \tag{16}$$

где x – независимая переменная.

Наивысший порядок производной n, входящей в уравнение (16), называют *порядком дифференциального уравнения*.

Из общей формы записи дифференциального уравнения (16) можно выразить производную в явном виде:

$$y' = f(x, y) \tag{17}$$

Уравнение (17) будет иметь бесконечное множество решений, поэтому для получения единственного решения необходимо указывать дополнительные условия.

Многие методы решения обыкновенных дифференциальных уравнений основаны на *задаче Коши*. Сформулируем эту задачу.

Пусть дано обыкновенное дифференциальное уравнение первого порядка, разрешенное относительно производной:

$$y' = f(x, y) \tag{18}$$

удовлетворяющее начальному условию:

$$y(x_0) = y_0 \tag{19}$$

Требуется найти на отрезке $[x_0, x_n]$ непрерывную функцию $y = \varphi(x)$, удовлетворяющую дифференциальному уравнению (18) и начальному условию (19), т.е. найти решение дифференциального уравнения. Поиск такого решения называется решением *задачи Коши*.

Численное решение этой задачи заключается в построении таблицы приближенных значений $y_1, y_2, \ldots, y_n$ (решения $y(x)$) в точках $x_1, x_2, \ldots, x_n$ с некоторым шагом h:

$$x_i = x_0 + i \cdot h, \qquad i = 1, 2, \ldots, n$$

Численные методы решения обыкновенных дифференциальных уравнений используются в тех случаях, когда нет возможности построить аналитическое решение задачи через элементарные функции.

Метод Эйлера

Метод Эйлера является наиболее простым численным методом решения обыкновенных дифференциальных уравнений и, как следствие, это достаточно грубый метод, однако его идеи легли в основу широкого класса численных методов.

Допустим, необходимо найти приближенное решение дифференциального уравнения первого порядка:

$$y' = f(x, y) \tag{20}$$

с начальным условием:

$$y(x_0) = y_0 \tag{21}$$

иначе говоря, необходимо решить задачу Коши.

Метод Эйлера

Разложим функцию $y(x)$ в окрестности точки x_0 в ряд Тейлора:

$$y(x) = y(x_0) + (x - x_0) y'(x_0) + \frac{(x - x_0)^2}{2} y''(x_0) + \dots \tag{22}$$

который применяется для приближенного определения значения искомой функции $y(x)$. В точке $x_0 + h$ при малых значениях h достаточно использовать только два слагаемых ряда (22), получим

$$y(x) = y(x_0 + h) = y(x_0) + y'(x_0) \Delta x + O(h^2) \tag{23}$$

где $O(h^2)$ – бесконечно малая величина порядка h^2. В формуле (23) сделаем следующие замены: производную функции $y'(x_0)$ заменим на правую часть уравнения (20); $\Delta x = h$; $y(x_0) = y_0$:

$$y(x_0 + h) \approx y_0 + hf(x_0, y_0) \tag{24}$$

Теперь приближенное решение в точке $x_1 = x_0 + h$ может быть вновь рассмотрено как начальное условие, следовательно, используя формулу (24), можно найти значение искомой функции в следующей точке $x_2 = x_1 + h$.
Таким образом, был получен простой алгоритм решения задачи Коши, называемый *методом Эйлера* или методом *ломаных*.

Метод Эйлера

Метод Эйлера может быть представлен в виде последовательного применения формул:

$$\begin{aligned} x_1 &= x_0 + h; \qquad & y_1 &= y_0 + hy_0' = y_0 + hf\left(x_0, y_0\right) \\ x_2 &= x_1 + h; \qquad & y_2 &= y_1 + hy_1' = y_1 + hf\left(x_1, y_1\right) \\ & & \ldots & \\ x_i &= x_{i-1} + h; \qquad & y_i &= y_{i-1} + y_{i-1}' = y_{i-1} + hf\left(x_{i-1}, y_{i-1}\right) \end{aligned} \tag{25}$$

В общем виде формула Эйлера записывается следующим образом:

$$y_i = y_{i-1} + hf\left(x_{i-1}, y_{i-1}\right); \qquad x_i = x_{i-1} + h \tag{26}$$

Метод Эйлера

Второе название – «метод ломаных» обусловлено графической интерпретацией данного метода. Искомая функция $y(x)$ заменяется ломаной линией с узлами в точках $x_0, x_1, \ldots, x_n$. Метод Эйлера характеризуется достаточно высокой погрешностью вычисления: $\Delta \approx O(h)$. В дополнение, данный метод во многих случаях оказывается неустойчивым – малая ошибка (к примеру, заложенная в исходных данных) накапливается с увеличением x. Вместе с тем, как показано на рисунке 6, точность метода Эйлера повышается при уменьшении размера шага вычислений h. Здесь также стоит отметить, что чрезмерно малое значение величины h приводит к снижению производительности вследствие увеличения количества вычислений: чем меньше шаг вычислений – тем больше итераций необходимо выполнить.

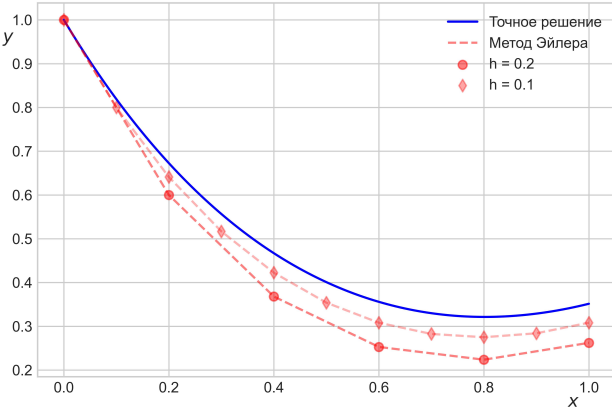

Рисунок 6 – Графическая интерпретация метода Эйлера

Пример

Рассмотрим решение обыкновенного дифференциального уравнения первого порядка:

$$\frac{dy}{dx} = \frac{y}{(\cos(x))^2}$$

методом Эйлера на отрезке $[0, 1]$ с шагом $h = 0.1$. Начальные условия: $x_0 = 0$; $y_0 = 2.7183$.

Используя формулу (26), построим таблицу значений переменной y_i при соответствующих значениях переменной x_i:

Пример

1. $y_1 = y_0 + h \cdot \frac{y_0}{(\cos(x_0))^2} = 2.7183 + 0.1 \cdot \frac{2.7183}{(\cos(0))^2} = 2.9901$
2. $y_2 = y_1 + h \cdot \frac{y_1}{(\cos(x_1))^2} = 2.9901 + 0.1 \cdot \frac{2.9901}{(\cos(0.1))^2} = 3.2922$
3. $y_3 = y_2 + h \cdot \frac{y_2}{(\cos(x_2))^2} = 3.2922 + 0.1 \cdot \frac{3.2922}{(\cos(0.2))^2} = 3.6349$
4. $y_4 = y_3 + h \cdot \frac{y_3}{(\cos(x_3))^2} = 3.6349 + 0.1 \cdot \frac{3.6349}{(\cos(0.3))^2} = 4.0332$
5. $y_5 = y_4 + h \cdot \frac{y_4}{(\cos(x_4))^2} = 4.0332 + 0.1 \cdot \frac{4.0332}{(\cos(0.4))^2} = 4.5086$
6. $y_6 = y_5 + h \cdot \frac{y_5}{(\cos(x_5))^2} = 4.5086 + 0.1 \cdot \frac{4.5086}{(\cos(0.5))^2} = 5.0940$
7. $y_7 = y_6 + h \cdot \frac{y_6}{(\cos(x_6))^2} = 5.0940 + 0.1 \cdot \frac{5.0940}{(\cos(0.6))^2} = 5.8418$
8. $y_8 = y_7 + h \cdot \frac{y_7}{(\cos(x_7))^2} = 5.8418 + 0.1 \cdot \frac{5.8418}{(\cos(0.7))^2} = 6.8404$
9. $y_9 = y_8 + h \cdot \frac{y_8}{(\cos(x_8))^2} = 6.8404 + 0.1 \cdot \frac{6.8404}{(\cos(0.8))^2} = 8.2497$
10. $y_{10} = y_9 + h \cdot \frac{y_9}{(\cos(x_9))^2} = 8.2497 + 0.1 \cdot \frac{8.2497}{(\cos(0.9))^2} = 10.3847$

Программная реализация


```python
import math


def func(x, y):
    return y / (math.cos(x) ** 2)


def eiler(func, x0, xf, y0, h):
    count = int((xf - x0) / h) + 1
    y = [y0]
    x = x0
    for i in range(1, count):
        y.append(y[i-1] + h * func(x, y[i-1]))
        x += h

    return y


print(eiler(func, 0, 1, 2.7183, 0.1))
#[2.7183, 2.99013, 3.2921531777519295, 3.6348964001721735,
 #4.033167969573622, 4.508579299929644, 5.093994160733677,
 #5.841814495589913, 6.840443308551035, 8.249681038793268, 10.384697446612579]

```

Метод Рунге-Кутты

Данный метод является широко используемым при интегрировании обыкновенных дифференциальных уравнений. Фактически речь пойдет о методе Рунге-Кутты четвертого порядка точности (существуют еще первого, второго и третьего порядка точности), из-за большой распространенности данного метода, указания на тип и порядок зачастую опускаются.

Классический метод Рунге-Кутты описывается следующим соотношением:

$$\begin{aligned} y_i &= y_{i-1} + \frac{h}{6} \cdot (k_1 + 2 \cdot k_2 + 2 \cdot k_3 + k_4) \\ x_i &= x_{i-1} + h \end{aligned} \tag{27}$$

где

$$\begin{aligned} k_1 &= f\left(x_{i-1}, y_{i-1}\right) \\ k_2 &= f\left(x_{i-1} + \frac{h}{2}, y_{i-1} + \frac{h}{2} \cdot k_1\right) \\ k_3 &= f\left(x_{i-1} + \frac{h}{2}, y_{i-1} + \frac{h}{2} \cdot k_2\right) \\ k_4 &= f\left(x_{i-1} + h, y_{i-1} + h \cdot k_3\right) \end{aligned} \tag{28}$$

Метод Рунге-Кутты

В сравнении с методом Эйлера, метод Рунге-Кутты дает результаты более близкие к точному решению даже при достаточно высоких значениях шага интегрирования.

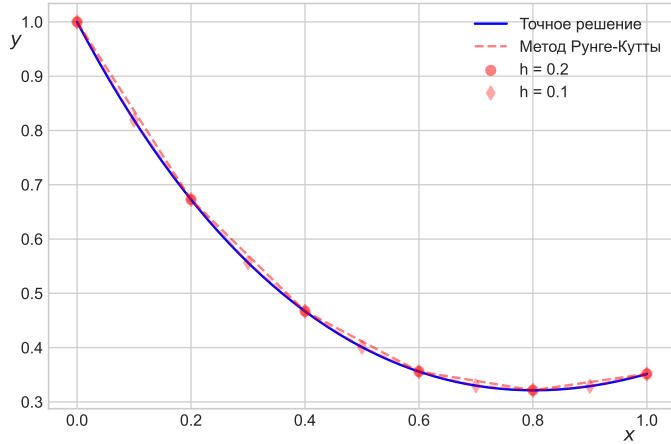

Рисунок 7 – Графическая интерпретация метода Рунге-Кутты

Пример

Рассмотрим решение примера, приведенного на слайде 34, методом Рунге-Кутты. Воспользуемся формулами (27), (28) и построим с их помощью таблицу искомых значений переменной y_i, соответствующих значениям переменной x из диапазона $[0, 1]$ с шагом $h = 0.1$. Подставим в формулу (28) выражение правой части исходного дифференциального уравнения и запишем формулы для определения параметров метода Рунге-Кутты:

$$k_1 = \frac{y_{i-1}}{(\cos(x_{i-1}))^2}$$

$$k_2 = \frac{y_{i-1} + \frac{h}{2} \cdot k_1}{\left(\cos\left(x_{i-1} + \frac{h}{2}\right)\right)^2}$$

$$k_3 = \frac{y_{i-1} + \frac{h}{2} \cdot k_2}{\left(\cos\left(x_{i-1} + \frac{h}{2}\right)\right)^2}$$

$$k_4 = \frac{y_{i-1} + h \cdot k_3}{(\cos(x_{i-1} + h))^2}$$

Пример

Результаты сведем в таблице:

x_i	k_1	k_2	k_3	k_4	y_i
0.0	–	–	–	–	2.7183
0.1	2.7183	2.8614	2.8685	3.0354	3.0052
0.2	3.0354	3.2291	3.2390	3.4659	3.3291
0.3	3.4659	3.7308	3.7449	4.0580	3.7037
0.4	4.0581	4.4272	4.4481	4.8901	4.1487
0.5	4.8903	5.4184	5.4509	6.0947	4.6941
0.6	6.0951	6.8779	6.9318	7.9088	5.3878
0.7	7.9096	9.1256	9.2215	10.7866	6.3110
0.8	10.7883	12.7957	12.9832	15.6764	7.6114
0.9	15.6806	19.2742	19.6867	24.7932	9.5846
1.0	24.8050	31.9927	33.0548	44.1554	12.9022

Программная реализация


```python
import math


def func(x, y):
    return y / (math.cos(x)) ** 2


def rk(func, x0, xf, y0, h):
    count = int((xf - x0) / h) + 1
    k1, k2, k3, k4 = 0, 0, 0, 0
    y = [y0]
    x = x0

    for i in range(1, count):
        k1 = func(x, y[i-1])
        k2 = func(x + h / 2, y[i-1] + h / 2 * k1)
        k3 = func(x + h / 2, y[i-1] + h / 2 * k2)
        k4 = func(x + h, y[i-1] + h * k3)
        y.append(y[i-1] + h / 6 * (k1 + 2 * k2 + 2 * k3 + k4))

        x += h

    return y

print(rk(func, 0, 1, 2.7183, 0.1))
#[2.7183, 3.0051916370697, 3.3291488983864, 3.70373913116160, 4.1487199630548, 4.6941126939828,
 #5.3878337155987, 6.3110044287492, 7.6113798673786, 9.584639573593, 12.9022298485301]

```

ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Контакты

Вячеслав Алексеевич Чузлов,
к.т.н., доцент ОХИ ИШПР

Учебный корпус №2, ауд. 136
chuva@tpu.ru
+7-962-782-66-15

Благодарю за внимание!