Научное програмирование

Лабораторная работа № 6

Кейела Патачона, НПМмд-02-21

Содержание

List of Figures

1 Цель работы

Вычисление пределов, сумм рядов и интегралов

2 Пределы, последовательности и ряды

Octave - полноценный язык программирования, поддерживающий множество типов циклов и условных операторов. Однако, поскольку то векторный язык, многие вещи, которые можно было бы сделать с помощью циклов, можно векторизовать. Под векторизованным кодом мы понимаем следующее: вместо того, чтобы писать цикл для многократной оценки функции, мы сгенерируем вектор входных значений, а затем оценим функцию с использованием векторного ввода. В результате получается код, который легче читать и понимать, и он выполняется быстрее благодаря эффективным алгоритмам для матричных операций.

2.1 Пределы

Рассмотрим предел:

$$\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n$$

Мы оценим это выражение. Сначала определим функцию. Есть несколько способов сделать это. Метод, который мы здесь используем, называется анонимной функцией. Это хороший способ быстро определить простую функцию.

```
>> diary on
>> f = @(n)(1 + 1. / n).^n
f =

@(n) (1 + 1. / n) .^ n

>> k = [0:1:9]'
k =

   0
   1
   2
   3
   4
   5
   6
   7
   8
   9

>> format long
>> n = 10 .^ k
n =

             1
            10
           100
          1000
         10000
        100000
       1000000
      10000000
     100000000
    1000000000

>> f(n)
```

Figure 2.1: Нахождение предела 1

Обращаем внимание на использование поэлементных операций. Мы назвали функцию f. Входная переменная обозначается знаком @, за которым следует переменная в скобках. Следующее выражение будет использоваться при оценке функции. Теперь f можно использовать как любую функцию в Octave. Далее мы создали индексную переменную, состоящую из целых чисел от 0 до 9 и взяли степени 10, которые будут входными значениями, а затем оценили $f(n)$.

```
>> f(n)
ans =

   2.000000000000000
   2.593742460100002
   2.704813829421529
   2.716923932235520
   2.718145926824356
   2.718268237197528
   2.718280469156428
   2.718281693980372
   2.718281786395798
   2.718282030814509

>> format
>>
```

Figure 2.2: Нахождение предела 2

Предел сходится к конечному значению, которое составляет приблизительно $2,71828\ldots$ Подобные методы могут быть использованы для численного исследования последовательностей и рядов.

2.2 Частичные суммы

Пусть $a = \sum_{n=2}^{\infty} a_n$ — ряд, n—й член равен

$$a_n = \frac{1}{n(n+2)}$$

Для этого мы определим индексный вектор пот 2 до 11, а затем вычислим члены. Если мы хотим знать частичную сумму, нам нужно только написать $sum(a)$. Если мы хотим получить последовательность частичных сумм, нам нужно использовать цикл. Мы будем использовать цикл for с индексом i от 1 до 10. Для каждого i мы получим частичную сумму последовательности a от первого слагаемого до i-го слагаемого. На выходе получается 10-элементный вектор этих частичных сумм.

```
>> n = [2:1:11]';
>> a = 1 ./ (n .* (n+2))
a =

   1.2500e-01
   6.6667e-02
   4.1667e-02
   2.8571e-02
   2.0833e-02
   1.5873e-02
   1.2500e-02
   1.0101e-02
   8.3333e-03
   6.9930e-03

>> for i = 1:10
s (i) = sum (a(1:i));
end
>> s'
ans =

   0.1250
   0.1917
   0.2333
   0.2619
   0.2827
   0.2986
   0.3111
   0.3212
   0.3295
   0.3365

>> plot(n,a,'o',n,s,'+')
>> grid on
>> legend('terms','partial sums')
>> print graph_01.png -dpng
```

Figure 2.3: Частичные суммы

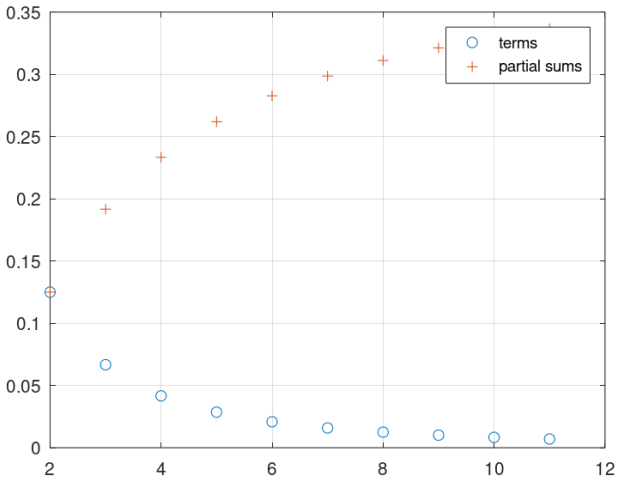

Figure 2.4: Частичные суммы - график

2.3 Сумма ряда

Найдём сумму первых 1000 членов гармонического ряда:

$$\sum_{n=1}^{1000} \frac{1}{n}$$

Нам нужно только сгенерировать члены как ряда вектор, а затем взять их сумму.

```
>>
>> n = [1:1:1000];
>> a = 1 ./ n;
>> sum(a)
ans = 7.4855
```

Figure 2.5: Сумма ряда

3 Численное интегрирование

3.1 Встроенная функция

Octave имеет несколько встроенных функций для вычисления определённых интегралов. Мы будем использовать команду $quad$ (сокращение от слова квадратура).

Вычислим интеграл:

$$\int_0^{\pi/2} e^{x^2} cos(x) dx$$

Синтаксис команды — $quadc('f', a, b)$. Нам нужно сначала определить функцию.

```
>>
>> function y = f(x)
y = exp (x .^ 2) .* cos (x);
end
>> quad ('f',0,pi/2)
ans = 1.8757
>>
>> f  = @(x) exp (x .^ 2) .* cos (x)
f =

@(x) exp (x .^ 2) .* cos (x)

>> quad (f,0,pi/2)
ans = 1.8757
>>
```

Figure 3.1: Определенный интеграл

Обращаем внимание, что функция $exp(x)$ используется для e. Мы использовали конструкцию $function...end$. Кавычки вокруг имени f не используются, если используется анонимная функция.

3.2 Правило средней точки

Правило средней точки, правило трапеции и правило Симпсона являются общими алгоритмами, используемыми для численного интегрирования. Напишем скрипт, чтобы вычислить интеграл

$$\int_0^{\pi/2} e^{x^2} cos(x) dx$$

по правилу средней точки для $n = 100$. Стратегия заключается в использовании цикла, который добавляет значение функции к промежуточной сумме с каждой итерацией. В конце сумма умножается на dx

```
% файл 'midpoint.m'
% вычисляет по правилу средней точти
% Интеграл от 0 до pi/2 от f(x) = exp(x^2)cos(x)

%Пределы игтегрирования
a = 0
b = pi / 2
n = 100 % Количество интевалов
dx = (b-a) / n

% Функция тнтегрирования
function y = f(x)
   y = exp(x .^ 2) .* cos(x);
end

% Цикл для расчета интеграла
msum = 0;
ml = a + dx / 2 ;% Первая точка
for i = 1:n
   m = ml + (i-1) * dx; % Средняя точка
   msum = msum + f(m);
end

% Аппроксимация
approx = msum * dx
```

Figure 3.2: Метод средней точки - код

```
>> midpoint
a = 0
b = 1.5708
n = 100
dx = 0.015708
approx = 1.8758
```

Figure 3.3: Метод средней точки - результат

Традиционный код работает хорошо, но поскольку Octave является векторным языком, также можно писать векторизованный код, который не требует каких-либо циклов. Создадим вектор r-координат средних точек. Затем мы оцениваем f по этому вектору средней точки, чтобы получить вектор значений функции. Аппроксимация средней точки – это сумма компонент вектора, умноженная на dx.

```
% файл 'midpoint_v.m'
% вычисляет по правилу средней точки
% Интеграл от 0 до pi/2 от f(x) = exp(x^2)cos(x)

%Пределы иггегрирования
a = 0
b = pi / 2
n = 100 % Количество интевалов
dx = (b-a) / n

% Функция тнтегрирования
function y = f(x)
  y = exp(x .^ 2) .* cos(x);
end

% Вектор точек
m = [a + dx/2 : dx : b-dx/2];

% Вектор значений
M = f(m);

% Аппроксимация
approx = dx * sum(M)
```

Figure 3.4: Векторный метод средней точки

```
>> midpoint_v
a = 0
b = 1.5708
n = 100
dx = 0.015708
approx = 1.8758
```

Figure 3.5: Векторный результат

3.3 Сравнение методов

Сравнили время выполнения для каждой реализации, и векторная реализация работает быстрее.

```
>> tic; midpoint; toc
a = 0
b = 1.5708
n = 100
dx = 0.015708
approx = 1.8758
Elapsed time is 0.0138249 seconds.
>>
>> tic; midpoint_v; toc
a = 0
b = 1.5708
n = 100
dx = 0.015708
approx = 1.8758
Elapsed time is 0.00774312 seconds.
>>
```

Figure 3.6: Сравнение реализаций

4 Вывод

В ходе выполнения данной работы мы ознакомились с вычислением пределов, с работой с последавательностями и с рядами и научились посчитать определенные интеграли с помощью различных методов на языке Octave.

Список литературы

1. Инструкция к лабораторной работе №6