Лабораторная работа №6

Дисциплина: Научное программирование

Аветисян Давид Артурович

Содержание

List of Tables

List of Figures

1 Цель работы

Познакомиться со сложными алгоритмами в Octave, которые были встроены для работы с пределами, последовательностями и рядами.

2 Задание

1. Познакомиться с пределами.
2. Познакомиться с частичными суммами.
3. Познакомиться с суммой ряда.
4. Познакомиться с вычислением интегралов.
5. Познакомиться с аппроксимированием суммами.

3 Выполнение лабораторной работы

1) Первым делом я научился работать с пределами. Для это я рассмотрел предел функции $f(n) = (1 + 1/n)^n$ при $n- > \infty$. Сначала я определил данную функцию и создал индексную переменую, состоящую из целых чисел от 0 до 9. Затем я задал длинный формат чисел.

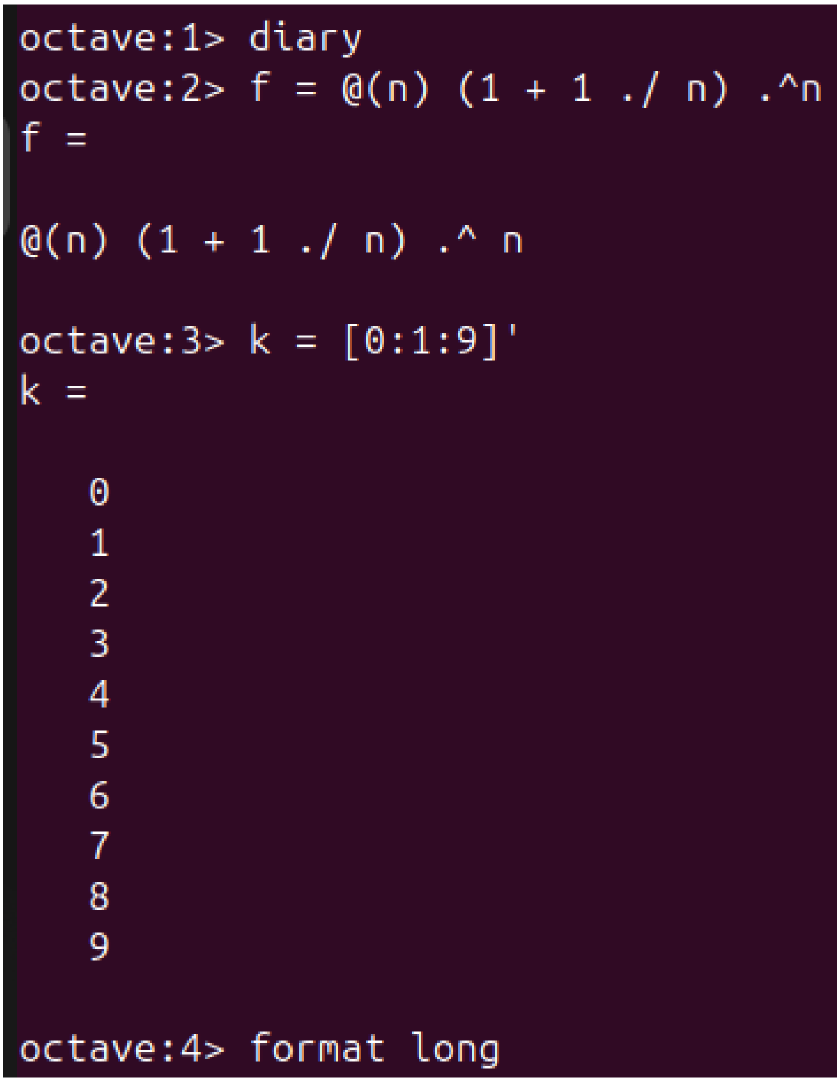


Figure 3.1: Определение функции и индексной переменной

После этого я взял степени числа 10, которые стали удобным входным значением и оценил $f(n)$. Предел сходится к конечному значению, которое приблизительно равно $2,71828$.

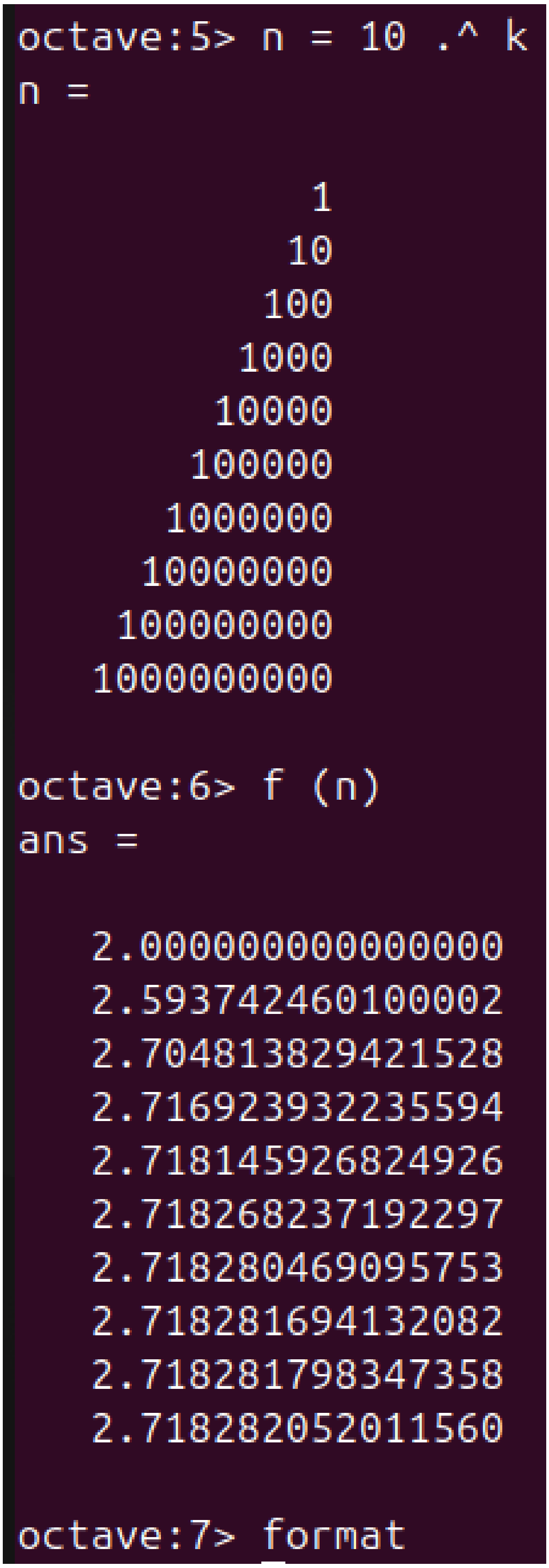
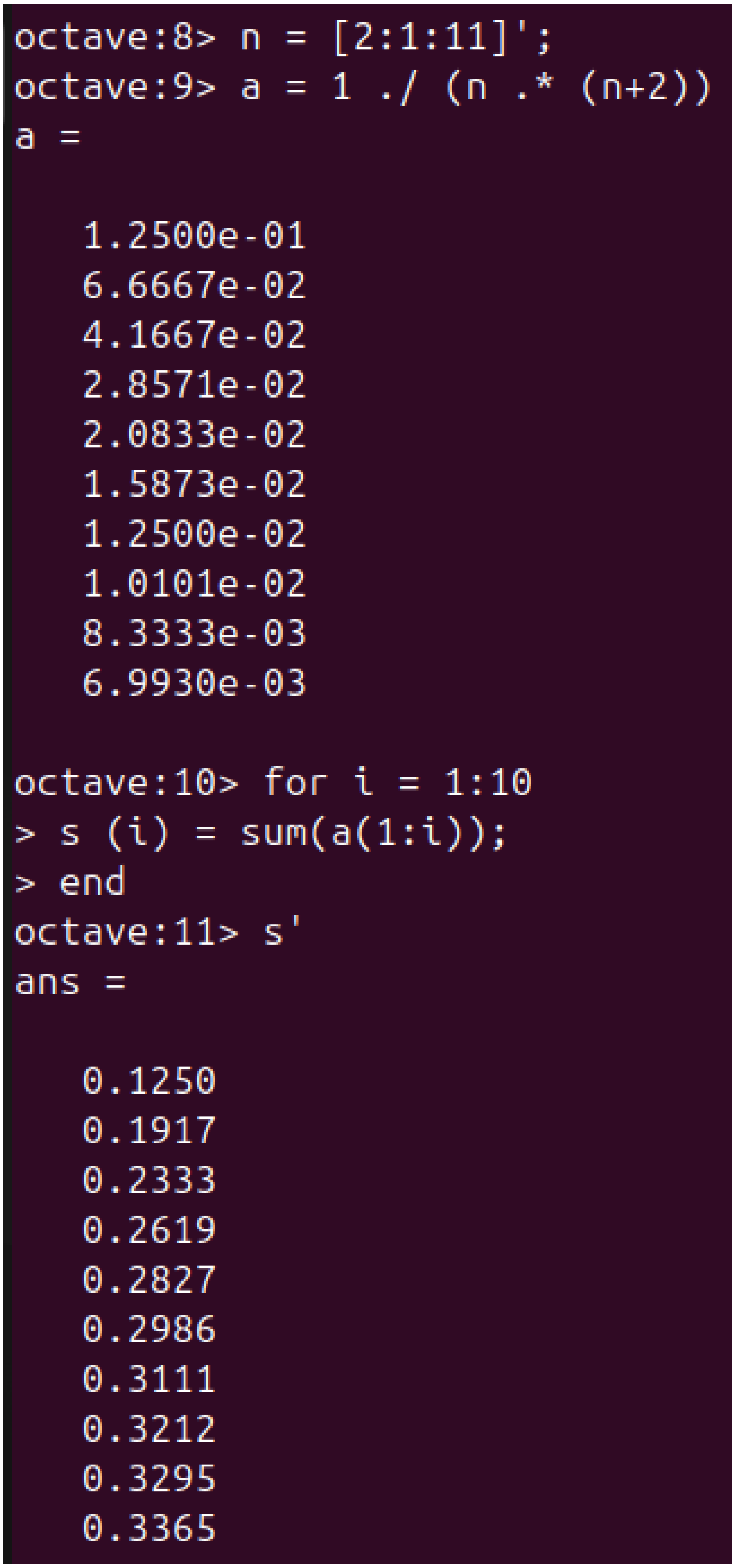


Figure 3.2: Оценка предела $f(n)$

2) Далее я познакомился с частичными суммами. Для этого я рассмотрел сумму ряда a от n при n от 2 до ∞, где $a = 1/(n * (n + 2))$. Сначала я определил индексный вектор n от 2 до 11. Затем я вычислил члены ряда a. После чего при помощи цикла я посчитал частичную сумму для каждого i ряда a от n от первого слагаемого до i-го слагаемого.



```
octave:8> n = [2:1:11]';
octave:9> 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

octave:10> for i = 1:10
> s (i) = sum(a(1:i));
> end
octave:11> s'
ans =

   0.1250
   0.1917
   0.2333
   0.2619
   0.2827
   0.2986
   0.3111
   0.3212
   0.3295
   0.3365
```

Figure 3.3: Определение частичной суммы ряда a

И в конце я построил слагаемые и частичные суммы для $2 <= n <= 11$.

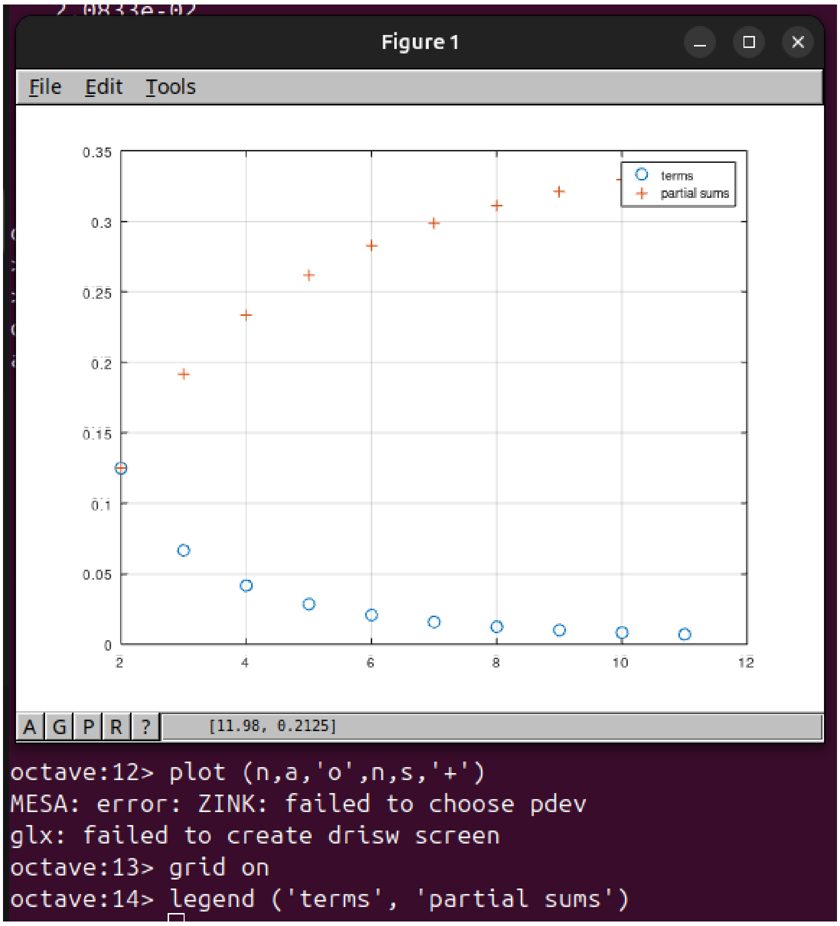


Figure 3.4: Построение слагаемых и частичных сумм для n

3) После я познакомился с нахождением суммы первых 1000 членов гармонического ряда $1/n$. Для этого я сгенерировал члены как вектор ряда, а затем взял их сумму.

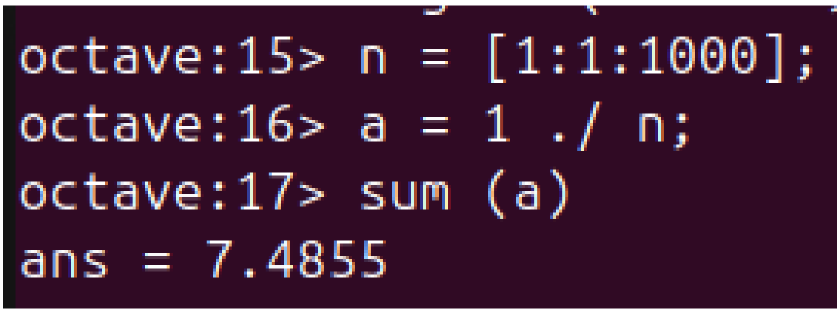
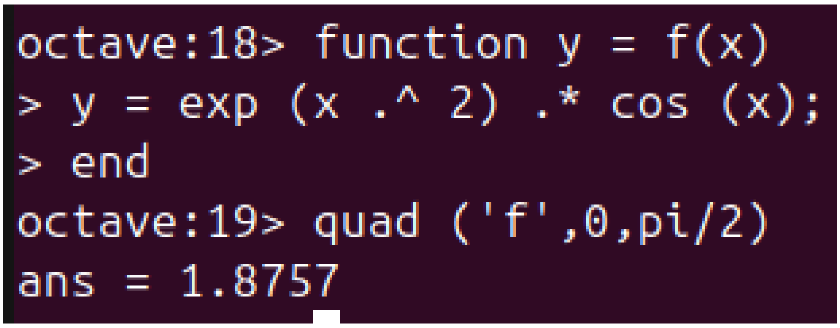


Figure 3.5: Нахождение суммы первых 1000 членов гармонического ряда $1/n$

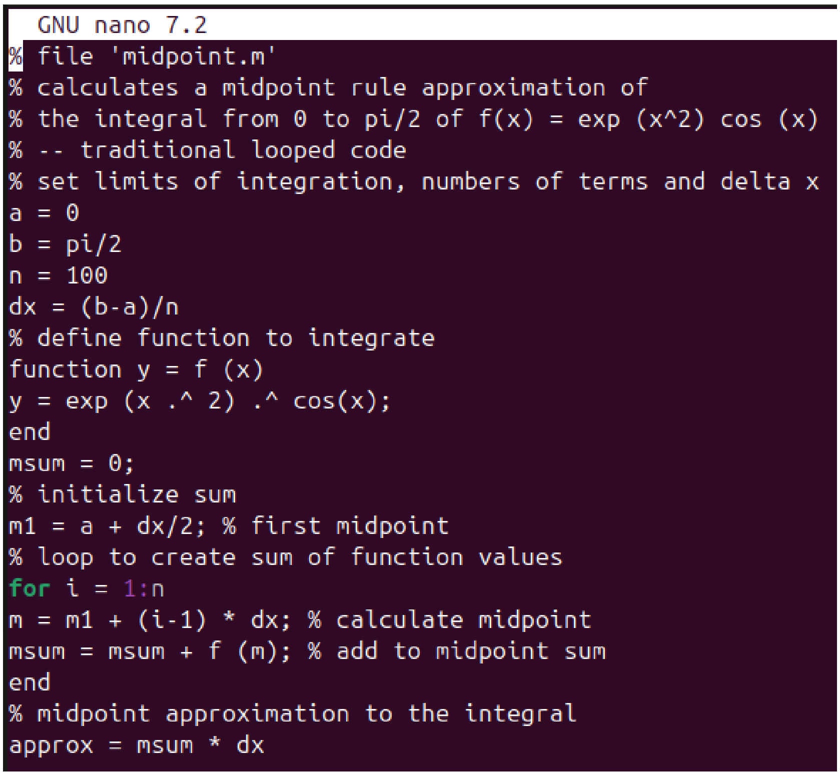
4) Потом я познакомился с вычислением интегралов. Для этого я взял интеграл от 0 до $pi/2$ функции $f(x) = exp(x^2) * cos(x)$. Для вычисления я определил функцию и использовал команду *quad*.



```
octave:18> function y = f(x)
> y = exp (x .^ 2) .* cos (x);
> end
octave:19> quad ('f',0,pi/2)
ans = 1.8757
```

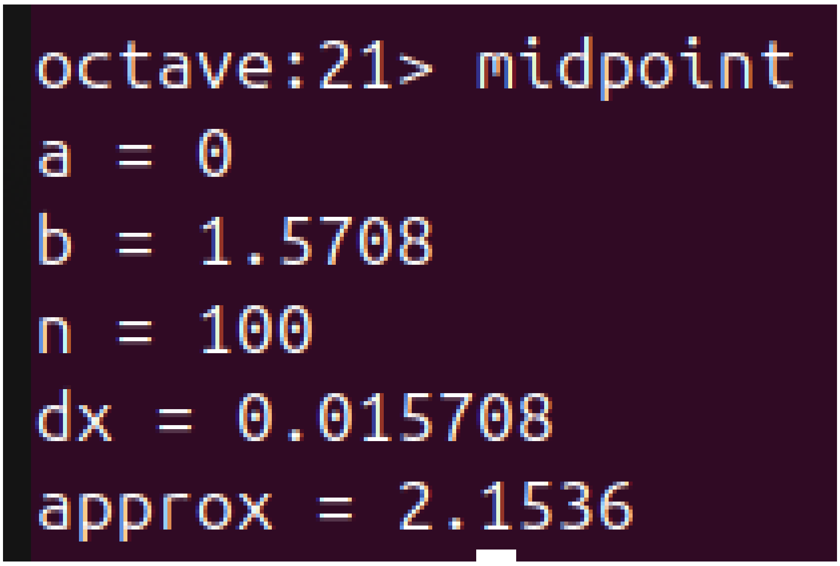
Figure 3.6: Вычисление интеграла $f(n)$

5) В конце лабораторой работы я научился аппроксимировать суммы. Сначала я написал скрипт чтобы вычислить предыдущий интеграл по правилу средней точки для $n = 100$ и запустил его.



```
  GNU nano 7.2
% file 'midpoint.m'
% calculates a midpoint rule approximation of
% the integral from 0 to pi/2 of f(x) = exp (x^2) cos (x)
% -- traditional looped code
% set limits of integration, numbers of terms and delta x
a = 0
b = pi/2
n = 100
dx = (b-a)/n
% define function to integrate
function y = f (x)
y = exp (x .^ 2) .^ cos(x);
end
msum = 0;
% initialize sum
m1 = a + dx/2; % first midpoint
% loop to create sum of function values
for i = 1:n
m = m1 + (i-1) * dx; % calculate midpoint
msum = msum + f (m); % add to midpoint sum
end
% midpoint approximation to the integral
approx = msum * dx
```

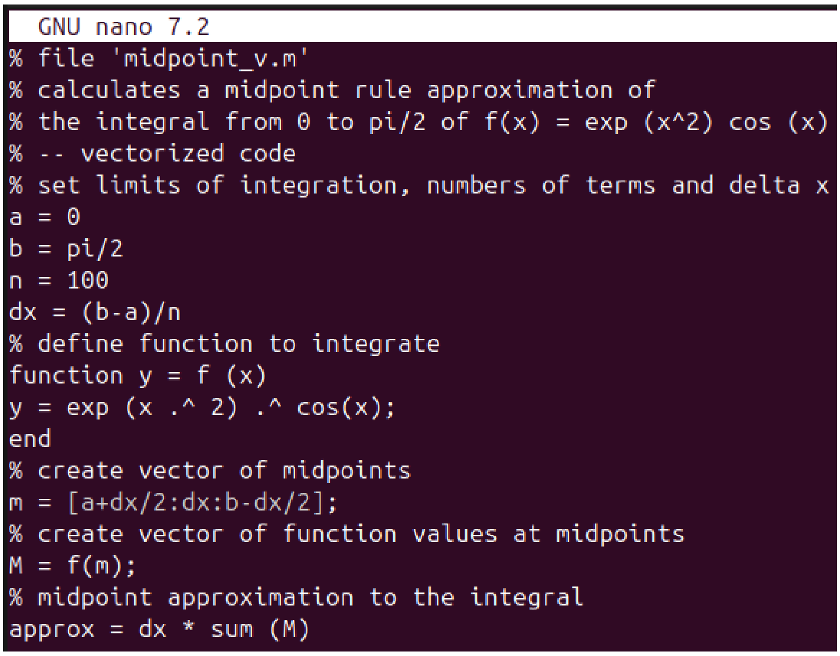
Figure 3.7: Скрипт midpoint.m



```
octave:21> midpoint
a = 0
b = 1.5708
n = 100
dx = 0.015708
approx = 2.1536
```

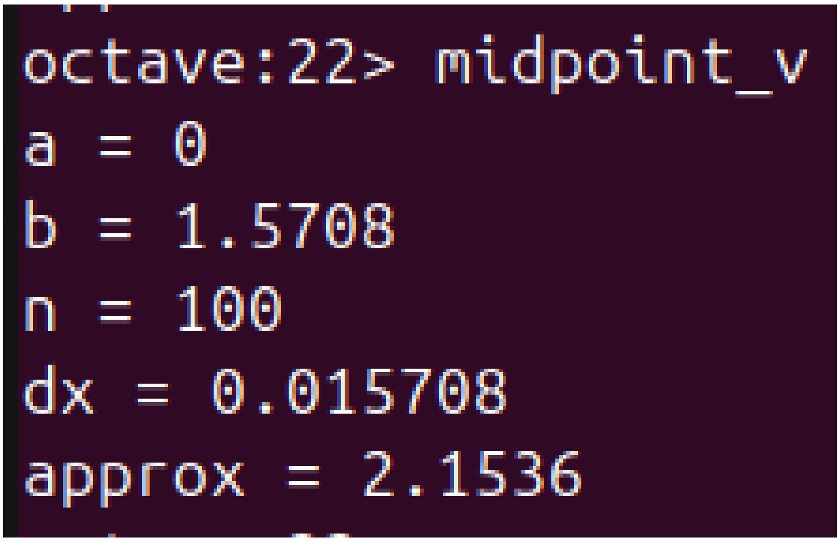
Figure 3.8: Работа скрипта midpoint.m

Далее я написал новый скрипт, но использовал векторизованный код, который не требует каких-либо циклов. После чего я его запустил.



```
  GNU nano 7.2
% file 'midpoint_v.m'
% calculates a midpoint rule approximation of
% the integral from 0 to pi/2 of f(x) = exp (x^2) cos (x)
% -- vectorized code
% set limits of integration, numbers of terms and delta x
a = 0
b = pi/2
n = 100
dx = (b-a)/n
% define function to integrate
function y = f (x)
y = exp (x .^ 2) .^ cos(x);
end
% create vector of midpoints
m = [a+dx/2:dx:b-dx/2];
% create vector of function values at midpoints
M = f(m);
% midpoint approximation to the integral
approx = dx * sum (M)
```

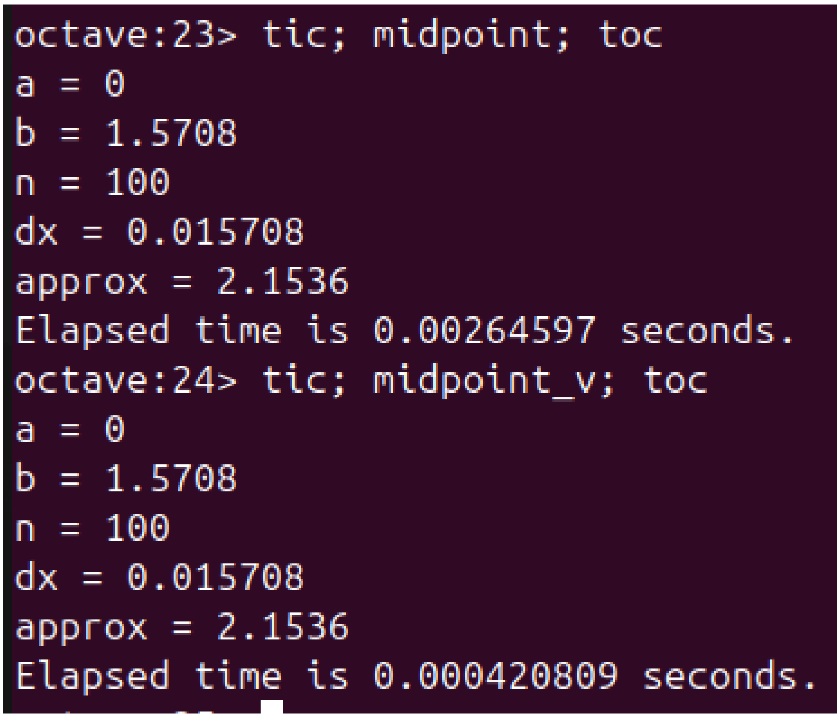
Figure 3.9: Скрипт midpoint_v.m



```
octave:22> midpoint_v
a = 0
b = 1.5708
n = 100
dx = 0.015708
approx = 2.1536
```

Figure 3.10: Работа скрипта midpoint_v.m

В конце я сравнил результаты и время выполнения каждого скрипта. Результаты были одинаковыми, но традиционный код выполнял почти в $6,5$ раз медленнее, чем векторизованный код.



```
octave:23> tic; midpoint; toc
a = 0
b = 1.5708
n = 100
dx = 0.015708
approx = 2.1536
Elapsed time is 0.00264597 seconds.
octave:24> tic; midpoint_v; toc
a = 0
b = 1.5708
n = 100
dx = 0.015708
approx = 2.1536
Elapsed time is 0.000420809 seconds.
```

Figure 3.11: Сравнение результатов и времени выполнения

4 Выводы

Я познакомился со сложными алгоритмами в Octave, которые были встроены для работы с пределами, последовательностями и рядами.