Белорусский государственный университет

Факультет прикладной математики и информатики

Кафедра технологий программирования

Стефанович Константин Андреевич

**Лабораторная работа №3**

**Метод Стеффенсена решения нелинейных уравнений**

студента 2 курса 6 группы

**Преподаватель**

***Радкевич Елена Владимировна*** Ассистент кафедры вычислительной математики ФПМИ

Минск, 2016

Оглавление

[1. Техническое задание 3](#_Toc476525872)

[2.Алгоритм решения и формулы 3](#_Toc476525873)

[3. Предварительные расчёты 4](#_Toc476525874)

[4. Листинг программы 4](#_Toc476525875)

[5.Результаты и вывод 4](#_Toc476525876)

# 1. Техническое задание

* Определить корни данного уравнения
* Выбрать
* Уравнение для работы:

# 2.Алгоритм решения и формулы

Необходимо ускорить скорость стремления ,

Если в формуле (1) , то при

(1)-даёт искомый предел. МПИ сходится по закону, близкому к показательному, (1) не будет давать при всяком n предельное значение, но приведёт к новой последовательности, сходящейся к искомому пределу более быстро.

По формуле (1) можем найти следующий эквивалент последовательности:

Заменим

(2) – Метод Стеффенсона.

# 3. Предварительные расчёты

1)Построим график нашей функции:

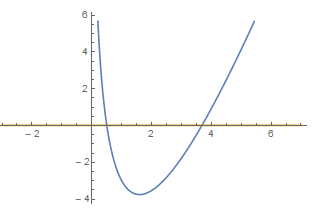


Рис.1. График

2) Приведём уравнение к каноническому виду:

3)Искомыми корнями будут x ≈ 0.504131735346712... и x ≈ 3.68823824651775... Будем искать второй корень.

4)Корень лежит в промежутке [3.5,3.7] . f(3.5)=-0.52 ; f(3.7)=0.0333, так что и условие выполняется

5)Возьмём

6) Тогда

7) Как видно, f дважды непрерывно диффиренцируема на

# 4. Листинг программы

//script.py

**from** numpy **import** \*  
**from** math **import** \*  
x0=3.7  
e=1e-4  
n=0  
**def** phi(x):  
 **return** (8/5)\*(log(x) + 1)  
**while** (**True**):  
 n+=1  
 x=(x0\*phi(phi(x0))-phi(x0)\*phi(x0))/(phi(phi(x0))-2\*phi(x0)+x0)  
 **if** abs(x0 - x) <= e:**break** x0=x  
print(**"За {0} итераций получили корень {1} с точностью {2}"**.format(n,x0,e))  
print(**"Вектор невязки:{0}"**.format(x0-(8/5)\*(log(x0)+1)))

# 5.Результаты и вывод

В результате работы программы получается следующий вывод:

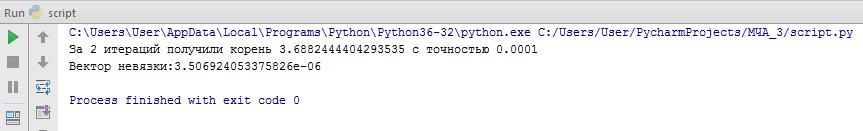


Рис.2. Окно вывода

Как видно, реальное количество итераций оказалось равным 2, что значительно лучше 7 итераций в методе простой итерации, но ровно столько же итераций получилось и в методе Ньютона