Университет ИТМО

Факультет программной инженерии и компьютерной техники

Дисциплина «Вычислительная математика»

Отчет

По лабораторной работе №2

Выполнил:

Терновский И.Е

Преподаватель:

Перл О.В

Санкт-Петербург, 2023 г.

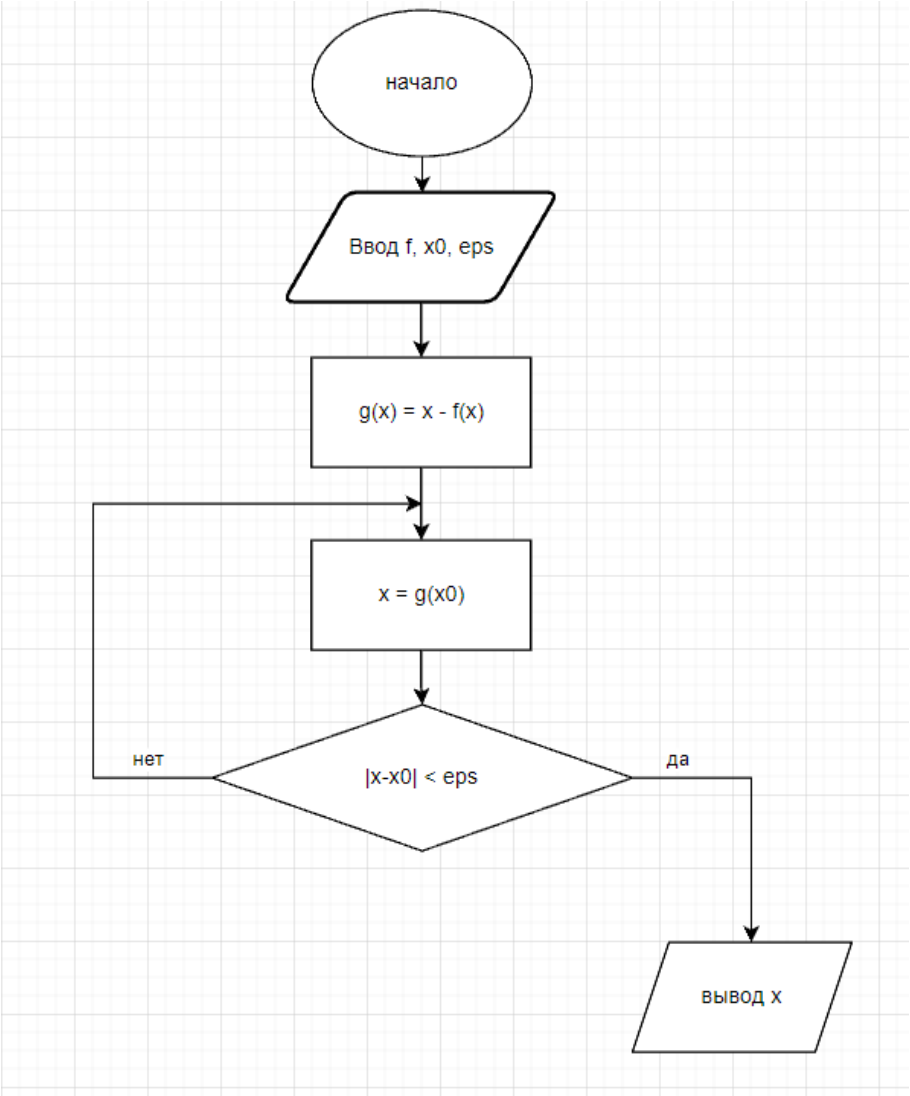
Описание методов

Метод простой итерации:

Метод основан на нахождении по приближенному значению величины следующее приближение. Метод позволяет получить решение с заданной точностью в виде последовательности итераций. То есть суть метода заключается в том, чтобы на каждом шаге получать все более точное решение.

При этом у метода есть некоторые особенности, главная заключатся в малой сходимости в окрестностях корня, поэтому для использования этого метода для начала следует получить приближенное значение корня.

Блок-схема:



Реализация на Python:

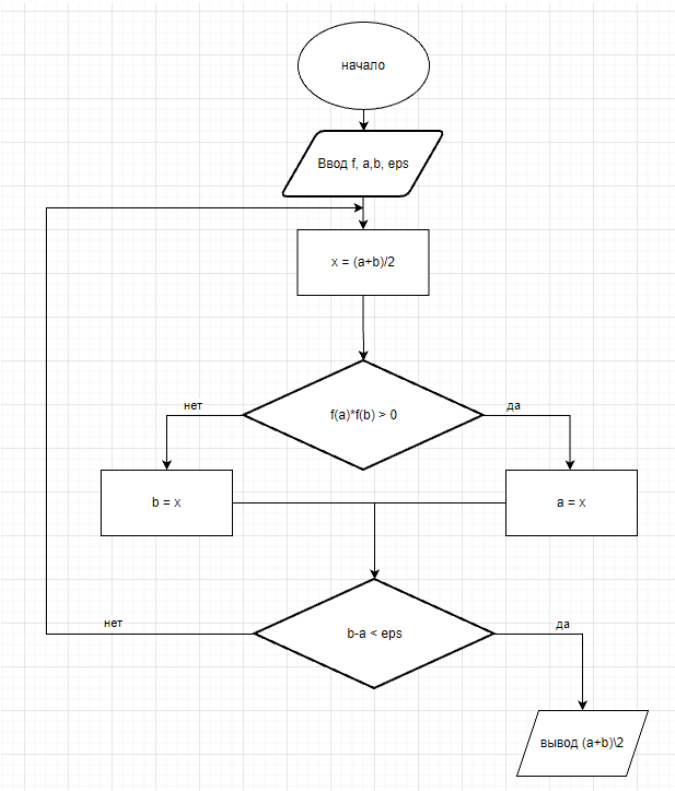
```python
def simple_iteration_method(g, x0=0.99, _eps=1e-5, max_iterations=1000):
    """
    Решает нелинейное уравнение методом простых итераций.

    g: преобразованная функция f, (в моем примере x-f(x)).
    x0: начальное приближение.
    _eps: заданная точность.
    max_iterations: максимальное количество итераций.
    """
    x_old = x0
    for i in range(max_iterations):
        x_new = g(x_old)
        if abs(x_new - x_old) < eps:
            return x_new
        x_old = x_new
    raise Exception("Превышено максимальное количество итераций.")
```

Метод деления пополам:

Суть метода основывается на теореме о промежуточных значениях, который заключатся в том, что на концах некоторого отрезка внутри которого есть корень уравнения буду разные знаки.

Собственно, для нахождения корня мы выбираем некоторый отрезок, выбираем у него середину и смотрим на каком из получившихся двух отрезков меняется знак, дальше работаем с ним.

Блок-схема:



Реализация на Python:

```
def bisection_method(f, a, b, _eps):
    """
    Решает нелинейное уравнение методом деления пополам.

    f: функция, содержащая неизвестное значение x.
    a, b: начальные границы для решения уравнения.
    _eps: заданная точность.
    """
    if f(a) * f(b) > 0:
        raise Exception("На отрезке нет корней")
    while abs(b - a) > _eps:
        c = (a + b) / 2
        if f(c) == 0:
            return c
        elif f(a) * f(c) < 0:
            b = c
        else:
            a = c
    return (a + b) / 2
```

Пример работы программы для этих двух методов:

1.

```
Введите 1 для решения уравнения и 2 для решения системы: 1
Выберите какое уравнение решать:
1.-(0.5 + x^2 - cos(x))
2.sin(x)
3.-(log(x)-cos(x))
1
Введите приближенное значение(для метода итераций): 1
Введите границы для метода деления пополам: 0 1
Введите желаемую точность: 0.001
Введите максимальное количество итераций: 100
Ответ методом простых итераций: 0.5797206879721211.
Ответ методом деления пополам: 0.57958984375.
Разница между этими методами:0.00013084422212106261.
```

2.

```
Введите 1 для решения уравнения и 2 для решения системы: 1
Выберите какое уравнение решать:
1.-(0.5 + x^2 - cos(x))
2.sin(x)
3.-(log(x)-cos(x))
2
Введите приближенное значение(для метода итераций): 1
Введите границы для метода деления пополам: 3 4
Введите желаемую точность: 0.00001
Введите максимальное количество итераций: 100
Ответ методом простых итераций: 3.141592653589793.
Ответ методом деления пополам: 3.141590118408203.
Разница между этими методами:2.535181589990998e-06.
```

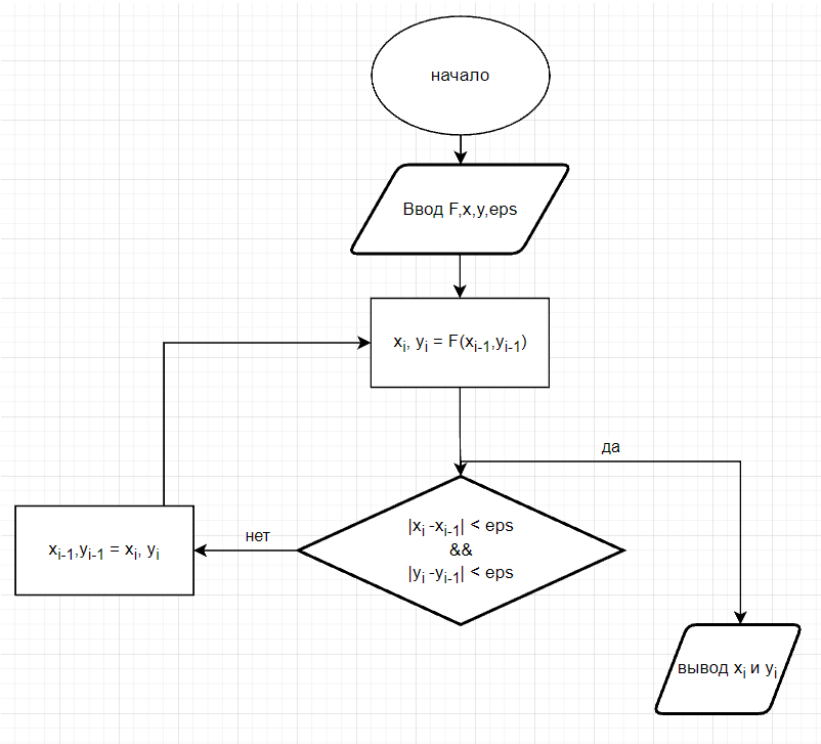
3.

```
Введите 1 для решения уравнения и 2 для решения системы: 1
Выберите какое уравнение решать:
1.-(0.5 + x^2 - cos(x))
2.sin(x)
3.-(log(x)-cos(x))
3
Введите приближенное значение(для метода итераций): 1
Введите границы для метода деления пополам: 1 2
Введите желаемую точность: 0.00001
Введите максимальное количество итераций: 100
Ответ методом простых итераций: 1.3029598997163414.
Ответ методом деления пополам: 1.3029670715332031.
Разница между этими методами:7.171816861717417e-06.
```

Метод итераций для систем нелинейных уравнений:

Метод итераций для решения системы нелинейных уравнений заключается в последовательном приближенном нахождении корней системы. Он основан на принципе, что если начать с некоторого приближенного решения системы, то можно получить новое приближение к решению, путем подстановки предыдущего решения в исходные уравнения. Этот процесс продолжается до тех пор, пока полученное приближение не удовлетворит заданной точности.

Блок-схема:



Пример работы:

```
Введите 1 для решения уравнения и 2 для решения системы: 2
Выберите какое уравнение решать:
4.1+cos(y)=0
1+sin(x)=0
5.x^2+y^2=1
1+sin(x)=0
4
Введите 2 приближенных значения x и y:0.8 1.7
Введите желаемую точность: 0.0001
Корни: x: 0.8711555057044753 y: 1.7173560908995227
```

Вывод

Проанализировав и реализовав некоторые методы для решения нелинейных уравнений, я понял, что есть множество разных методов для решения этих уравнений и что каждый из методов хорош по-своему и применяется в определенных случаях. Некоторые могут быть более точными, некоторые более быстрыми, некоторые возможно быстрее реализовать, но самое главное что я понял это то что реализовывать их самим не надо, ведь есть прекрасная библиотека numpy которая все сделает сама.