Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №3
по дисциплине “Вычислительная математика”
Вариант: Метод простых итераций

Выполнила:
Шевченко Д. П.,
группа P3230

Преподаватель:
Перл О.В.

Санкт-Петербург
2024

Оглавление

Описание численного метода

Задана система нелинейных уравнений. Для решения приведем систему к эквивалентному виду:

$$\begin{cases} f_1(x_1, \dots, x_n) = 0 \\ \dots \\ f_n(x_1, \dots, x_n) = 0 \end{cases} \rightarrow \begin{cases} x_1 = \Phi_1(x_1, \dots, x_n) \\ \dots \\ x_n = \Phi_n(x_1, \dots, x_n) \end{cases}$$

После этого выбираем начальное приближение $x^{(0)} = (x_1^{(0)}, \dots, x_n^{(0)})$
Производим итерационный процесс, то есть высчитываем следующие приближения по формуле:
$x_i^{k+1} = \Phi_i(x_1^k, \dots, x_n^k)$, то есть:

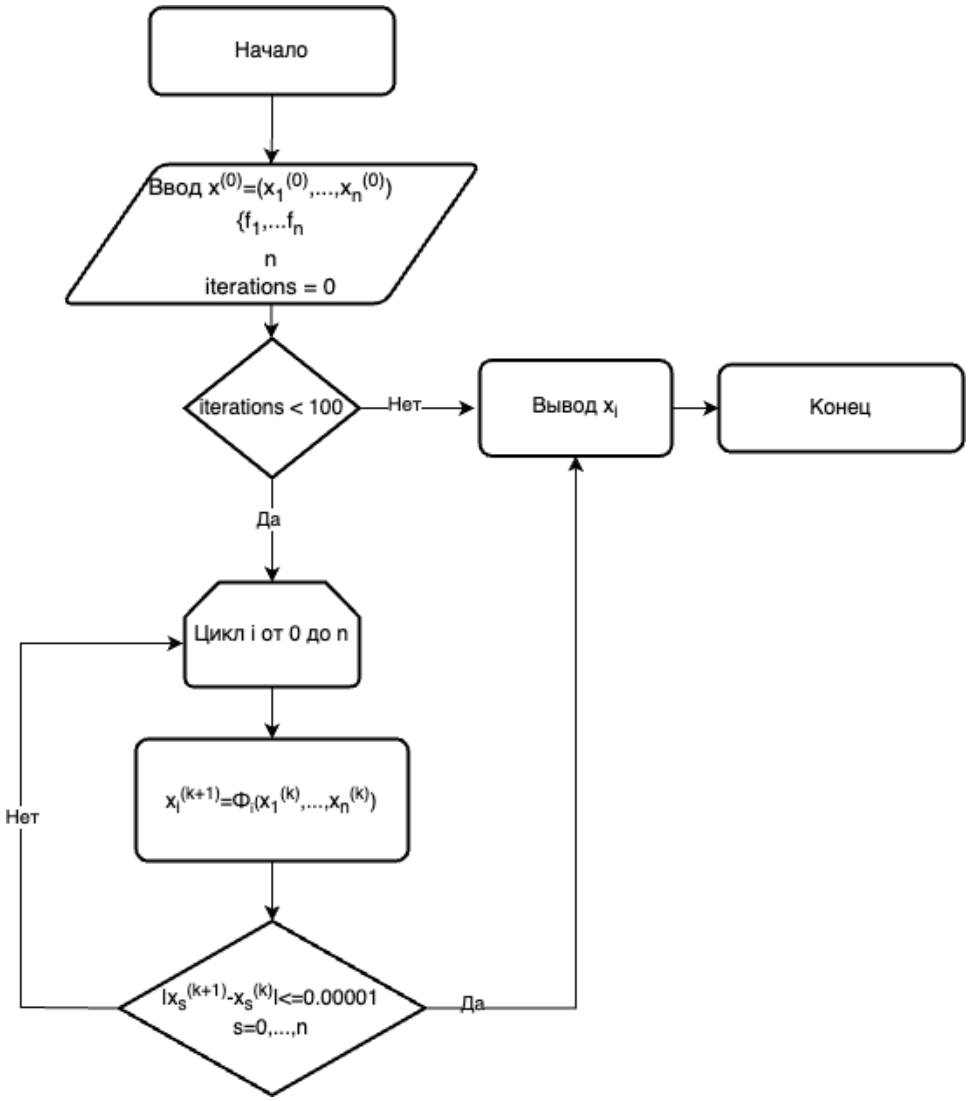
$$\begin{cases} x_1^{(k+1)} = \Phi_1\left(x_1^{(k)}, \dots, x_n^{(k)}\right) \\ \dots \\ x_n^{(k+1)} = \Phi_n\left(x_1^{(k)}, \dots, x_n^{(k)}\right) \end{cases}$$

Если итерационный процесс сходится и Ф(x) непрерывна, то мы найдем

$$\lim_{n\to\infty} x_n = \Phi(\lim_{n\to\infty} x_{n-1})$$

Проверяем условие окончания $\left|x^{(k+1)} - x^{(k)}\right| \leq e$, либо заканчиваем, когда количество итераций дойдет до максимального числа.

Блок схема


Начало
Ввод $x^{(0)}=(x_1^{(0)},...,x_n^{(0)})$
$\{f_1,...f_n$
n
iterations = 0
iterations < 100
Нет
Вывод x_i
Конец
Да
Цикл i от 0 до n
$x_i^{(k+1)}=\Phi_i(x_1^{(k)},...,x_n^{(k)})$
Нет
$|x_s^{(k+1)}-x_s^{(k)}|<=0.00001$
s=0,...,n
Да

Код

```
def solve_by_fixed_point_iterations(system_id, number_of_unknowns,
initial_approximations):
    new_x = [0] * number_of_unknowns
    max_count_iterations = 100
    count_iteration = 0
    old_x = initial_approximations
    while count_iteration < max_count_iterations:
        check = 0
        count_iteration += 1
        try:
            new_x = [old_x[i] - 0.0001 * get_functions(system_id)[i](old_x)
for i in range(number_of_unknowns)]
        except IndexError:
            print("Недостаточно переменных")
            break
        for i in range(number_of_unknowns):
            if abs(new_x[i] - old_x[i]) < 0.00001:
                check += 1
        if check == number_of_unknowns:
            return new_x
        old_x = new_x
    return new_x
```

Примеры работы программы

1) Корректный ввод

```
2
2
-2
3
-0.6118481851016625
3.202022196290775
```

2) Недостаточное количество переменных, во всех функциях кроме первой необходимое количество неизвестных – две (в 5,6,7 функциях – три).

```
2
1
2
Недостаточно переменных
0
```

3) Корректный вывод для третьей системы

```
3
3
4
5
7
4.368164417066925
3.927898642834979
6.132967954733627
```

4) Корректный вывод для первой системы

```
1
2
4
1
4.007592492201102
0.9801783100312664
```

5) Слишком большое значение

```
2
2
4
7
Traceback (most recent call last):
  File "/Users/dasha/PycharmProjects/pythonProject4/main.py", line 100, in <module>
    result = solve_by_fixed_point_iterations(system_id, number_of_unknowns, initial_approximations)
  File "/Users/dasha/PycharmProjects/pythonProject4/main.py", line 76, in solve_by_fixed_point_iterations
    new_x = [old_x[i] - 0.0001 * get_functions(system_id)[i](old_x) for i in range(number_of_unknowns)]
  File "/Users/dasha/PycharmProjects/pythonProject4/main.py", line 76, in <listcomp>
    new_x = [old_x[i] - 0.0001 * get_functions(system_id)[i](old_x) for i in range(number_of_unknowns)]
  File "/Users/dasha/PycharmProjects/pythonProject4/main.py", line 19, in third_function
    return pow(args[0], 2) * pow(args[1], 2) - 3 * pow(args[0], 3) - 6 * pow(args[1], 3) + 8
OverflowError: (34, 'Result too large')
```

Выводы

1) Результаты запуска на странице 6
2) Сравнение с другими методами:

В сравнении с методом Ньютона, метод простых итераций сходится медленнее, что значительно увеличивает количество производимых итераций. Но метод простых итераций более легок в реализации, так как не требует вычисления производных. Стоит заметить, если метод Ньютона применется не вблизи точки решения, то сходится он будет дольше.

3) Анализ применимости метода:

Метод простых итераций подходит для простых систем нелинейных уравнений, в которых выполняются условия сходимости и выбрано верное начальное приближение. В случае более сложных систем или если требуется высокая скорость сходимости стоит выбрать другой метод, так как данный метод будет менее быстрым.

4) Анализ алгоритмической сложности:

В каждой итерации нужно вычислить новое приближение. Вычисляются новые значения переменных на основе предыдущих значений.
Сложность метода равна O(n*k), так как мы проходимся k (количество итераций) раз по n переменных

5) Анализ численной ошибки:

Ошибка зависит от количества итераций, если их будет недостаточно, то результат будет недостаточно приближен к правильному. При слишком большом количестве потребуется большой объем вычислений.
Погрешность можно вычислить по формуле

$$\varepsilon^{(k+1)} = \varepsilon^{(k)} * \frac{\partial \Phi(x)}{\partial x}$$

Метод простых итераций прост в реализаци. Если возникают неточности на какой-либо итерации, то она не отразится на конечном результате, а лишь на количестве итераций. Метод позволяет быстро и с малым количеством итераций достигнуть заданной точности, если начальное приближение выбрано вблизи корня, иначе объем вычислений значительно возрастет.