Университет ИТМО

Факультет программной инженерии и компьютерной техники

Дисциплина «Вычислительная математика»

Отчет

По лабораторной работе №3 (Метод Ньютона)

Выполнил:

Терновский И.Е

Преподаватель:

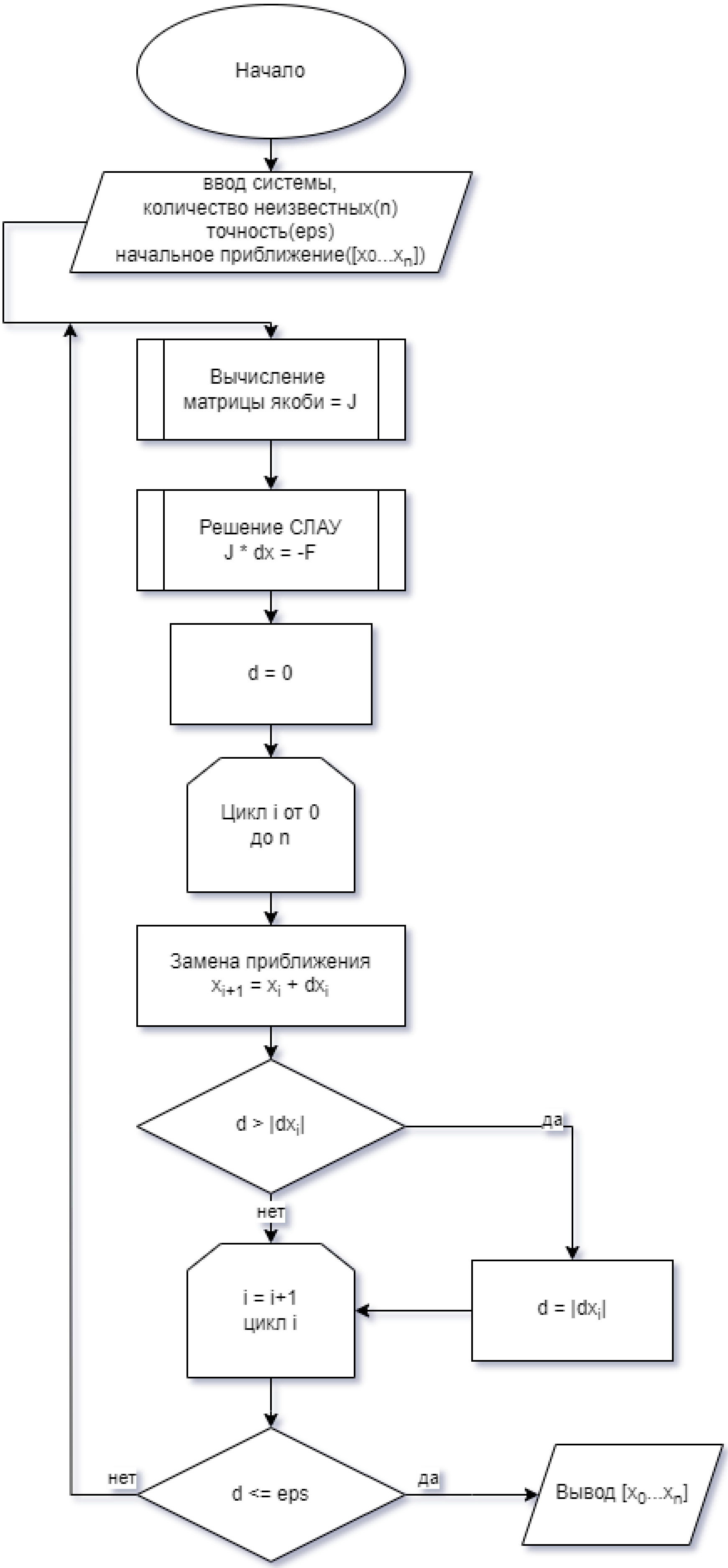
Перл О.В

Санкт-Петербург, 2024 г

Описание метода

Метод Ньютона для решения СНАУ это итерационный метод, основная идея которого заключается в замене СНАУ на СЛАУ, решение которой проще и может дать приближённый ответ для СНАУ. То есть в методе мы выбираем некоторое начальное приближение, затем производим линеаризацию для получения СЛАУ, решаем СЛАУ любым возможным методом и заменяем полученными значениями начальное приближение, выполняя данные действия мы каждый раз приближаемся к верному решению СНАУ.

Блок схема:


Начало
ввод системы,
количество неизвестных(n)
точность(eps)
начальное приближение([xo...x_n])
Вычисление
матрицы якоби = J
Решение СЛАУ
J * dx = -F
d = 0
Цикл i от 0
до n
Замена приближения
$x_{i+1} = x_i + dx_i$
d > $|dx_i|$
да
нет
i = i+1
цикл i
d = $|dx_i|$
нет
d <= eps
да
Вывод [x_0...x_n]

Функция численного метода:

```
def solve_by_fixed_point_iterations(system_id, number_of_unknowns,
initial_approximations, max_iterations):
    functions = get_functions(system_id)

    def find_jacobian(functions, variables):
        jacobian_matrix = [[0] * number_of_unknowns for _ in
range(number_of_unknowns)]
        for i in range(number_of_unknowns):
            for j in range(number_of_unknowns):
                small_change = [0] * number_of_unknowns
                small_change[j] = 1e-6
                jacobian_matrix[i][j] = (functions[i]([variables[dim] +
small_change[dim] for dim in range(number_of_unknowns)]) - functions[i](variables)) /
1e-6
        return jacobian_matrix

    def normalize(function_values):
        return math.sqrt(sum(value ** 2 for value in function_values))

    def subtract_vectors(vector_a, vector_b):
        return [a_i - b_i for a_i, b_i in zip(vector_a, vector_b)]

    def solve_linear_system(jacobian_matrix, function_values):
        number_of_variables = len(jacobian_matrix)
        augmented_matrix = jacobian_matrix

        # Append function_values to the matrix augmented_matrix
        for i in range(number_of_variables):
            augmented_matrix[i].append(function_values[i])

        # Gaussian elimination
        for i in range(number_of_variables):
            for j in range(i + 1, number_of_variables):
                ratio = augmented_matrix[j][i] / augmented_matrix[i][i]
                for k in range(number_of_variables + 1):
                    augmented_matrix[j][k] -= ratio * augmented_matrix[i][k]

        # Back substitution
        solutions = [0 for _ in range(number_of_variables)]
        solutions[number_of_variables - 1] = (augmented_matrix[number_of_variables -
1][number_of_variables] /
                                              augmented_matrix[number_of_variables -
1][number_of_variables - 1])
        for i in range(number_of_variables - 2, -1, -1):
            solutions[i] = augmented_matrix[i][number_of_variables]
            for j in range(i + 1, number_of_variables):
                solutions[i] -= augmented_matrix[i][j] * solutions[j]
            solutions[i] /= augmented_matrix[i][i]
        return solutions

    variables = initial_approximations
    for i in range(max_iterations):
        function_values = [function(variables) for function in functions]
        if normalize(function_values) < 1e-6:
            return variables
        jacobian_matrix = find_jacobian(functions, variables)
        for i in range(len(jacobian_matrix)):
            jacobian_matrix[i][i] += 1e-6
        variables = subtract_vectors(variables, solve_linear_system(jacobian_matrix,
function_values))

    raise RuntimeError("Метод Ньютона не сошелся за {}
итераций".format(max_iterations))
```

Пример работы:

1)
```
1
2
0
0
Ответ: 0.0 0.0
```
В данном примере используется система $sin(x) = 0$ и $(x * y)/2 = 0$ которая при приближении 0, 0 имеет корень в точке (0,0)

2)
```
1
2
2
0
Ответ: 3.1415926543296404 0.0
```
В данном примере используется та же система что и в п.1, но так как начальное приближение другое, корень получился другой $(\pi, 0)$

3)
```
2
2
1
1
Ответ: 0.9281401594896153 0.3351868755348791
```
В данном примере система состоит из уравнений $tan(xy + 0.4) - x^2$ и $0.9x^2 + 2y^2 - 1$ и при данном приближении метод сходится к правильному корню

4)
```
4
3
1
1
1
Ответ: 0.7851969521579405 0.49661139325580467 0.36992283089598216
```
В данном примере система состоит из 3 уравнений и сходится к верному корню

5)
```
5
2
1
1
Произошла ошибка, перепроверьте что система имеет решение: Метод Ньютона не сошелся за 1000 итераций
```
В данном примере система состоит из уравнений $x + y$-3 и $x + y -$ 2 и конечно не имеет решений, поэтому программа возвращает сообщение об ошибке

Вывод:

Метод Ньютона для решения СНАУ представляет собой сильный инструмент для нахождения корней СНАУ. В тоже время он достаточно сложный в реализации, так как в нем нужно находить якобиан и решать СЛАУ. В связи с чем его нельзя назвать очень эффективным, так как опять же на каждой итерации нам требуется находить обратный якобиан в связи с чем

алгоритмическая сложность данного алгоритма не самая лучшая, а именно в моем случае, когда для решения СЛАУ используется метод Гаусса, это $\boldsymbol{n^3}$, если же упросить вычисление СЛАУ, то сложность будет упираться в нахождение Якобиана, что занимает $\boldsymbol{n^2}$. Метод Ньютона может быть полезен когда система имеет не очень много неизвестных и когда у нас есть достаточно хорошее изначальное приближение, тогда за счет быстрой сходимости (квадратичной), мы можем достаточно быстро и точно найти корни. В сравнении с другими методами, этот является сложным в реализации и требует изначальное приближение, но при этом очень быстро сходится. Метод простой итерации к примеру не требует начального приближения и очень прост в реализации, но сходится он медленее(линейно). Так же метод Ньютона является родителем для усовершенствованных методов Ньютона, которые решают некоторые проблемы оригинального метода.
Численная ошибка как и для любого численного метода складывается из за переполнения ячеек памяти и округления.