Реализация итерационного метода решения нелинейных СЛАУ (2.3.3(а2) – метод секущих: простейшие аппроксимации частных производных)

Группа: ПМ-2001

Студент: Иксанов Марат Васильевич

1. Суть метода и алгоритм решения:

Дано уравнение $\mathbf{f}(\mathbf{x}) = \mathbf{0} \Leftrightarrow \begin{cases} f_1(x_1, \dots, x_s) = 0 \\ \dots \dots \dots \\ f_s(x_1, \dots, x_s) = 0 \end{cases}$

Искомым решением является вектор $x = (x_1, \dots, x_s)^T$, который строится с помощью итерационного метода.

Данный метод является дискретной модификацией метода Ньютона-Рафсона, в котором каждое новое итерационное решение $x_{k+1} = x_k - f_k'^{-1} f_k$, где $x_0 -$ начальное приближение, $f_k = f(x_k),\ f_k'^{-1} = f'^{-1}(x_k)$

Особенностью метода секущих является представление f_k' приблизительно равно $H_k^{-1}\Gamma_k$, а, следовательно, $f_k'^{-1}$ приблизительно равно $\Gamma_k^{-1} H_k$, где матрица $H_k -$ диагональная матрица приращений аргумента, каждый элемент h_{ii} можно принять за разность x_{k+1} и x_k $+$ некоторое малая величина, чтобы избежать случая вырожденной матрицы

Матрица Γ_k представляется матрицей приращений функций по каждой переменной, То есть $\gamma_{ij} = f_i(x_k + h_j e_j) - f_i(x_k)$, где $e_j -$ j столбец единичной матрицы размера sxs

Таким образом, конечная формула нахождения следующего итерационного приближения: $x_{k+1} = x_k - \Gamma_k^{-1} H_k f_k$, где на каждом следующем шаге, Γ_k, H_k, f_k повторно вычисляются

Существуют 3 основных критерия прекращения итерационного процесса:

1) По числу итераций. $k < K_{max}$, где K_{max} задается пользователем
2) По близости к решению нормы разности $x_{k+1} - x_k$, которая должна быть меньше заданного δ
3) По малости нормы значения функции в текущем итерационном методе. Такая норма должна быть меньше некоторого ε, которое задается пользователем

Для корректности работы стоит использовать как минимум два критерия прекращения итерационного процесса. Мною выбран 1 и 3 критерий.

2. Программная реализация:

Входные данные: система уравнений $\begin{cases} f_1(x_1, \dots, x_s) = 0 \\ \dots \dots \dots \\ f_s(x_1, \dots, x_s) = 0 \end{cases}$ и начальное приближение x_0

Выходные данные: найденный вектор $x = (x_1, \dots, x_s)^T$.

Метод секущих: простейшие аппроксимации частных производных реализован с помощью Wolfram Mathematica:

```
p = {};
e = 0.00000001;
k = 0;
kmax = 600;
While[k ≤ kmax ∧ Norm@f[Sequence @@ Flatten@x0] ≥ e,
 AppendTo[p, Flatten@x0];
 jacob = Table[D[f[q, w], i], {i, {q, w}}];
 jInv = Inverse[jacob] /. Thread[Rule[{q, w}, Flatten@x0]];
 x1 = x0 - jInv.f[Sequence @@ Flatten@x0];
 x0 = x1;
 k++;

]
```

Параметр $kmax$ задается пользователем и определяет максимальное количество итераций. Параметр e также задается пользователем и определяет число, норма значения исходной функции от текущего итерационного решения которого должна быть меньше этого числа, чтобы остановить итерационный процес.

3. Результат вычисления на предоставленных входных данных и оценка точности решения:

Для оценки точности решения рассмотрим входные данные, тип данных входящих систем – матрица, где каждая строка - нелинейное уравнение.

Входные данные:

$$f(x) = \begin{pmatrix} x_1^2 - x_2^2 - 1 \\ x_1 x_2^3 - x_2 - 1 \end{pmatrix}, x_0 = \begin{pmatrix} 1.5 \\ 1.5 \end{pmatrix}$$

Рассмотрим полученное решение с помощью написанной мною функции и сравним ее точность со встроенной функцией.

1. Собственная реализация при $kmax = 600, \varepsilon = 1.\times 10^{-18}$

Полученные данные:

```
Flatten@x0
```

```
{1.50284, 1.12185}
```

```
Norm@f[Sequence @@ Flatten@x0]
```

$$8.00593 \times 10^{-16}$$

Решение с помощью встроенной функции $NSolve$:

```
x = {q, w} /. NSolve[f[q, w] == 0, {q, w}, Reals] // First
```

```
{1.50284, 1.12185}
```

```
Norm@f[Sequence @@ Flatten@x]
```

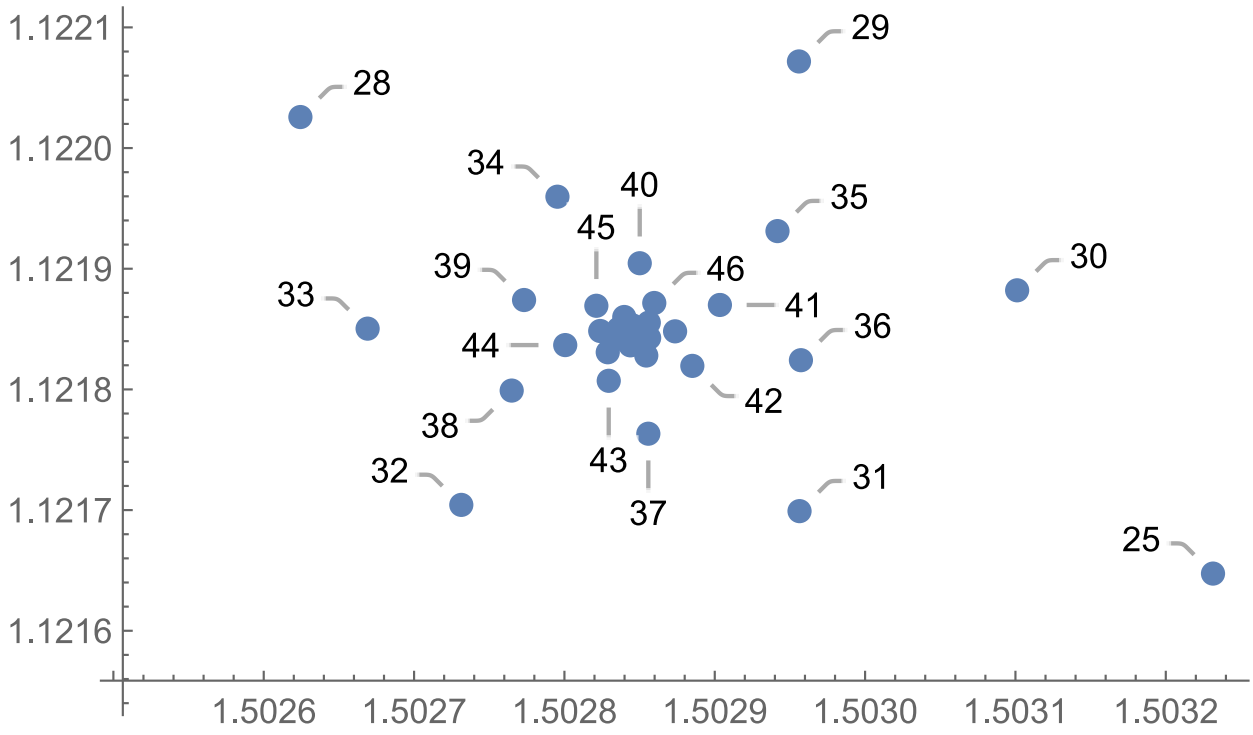
$$3.33067 \times 10^{-15}$$

Нормы векторов невязки практически не отличаются, что означает, что написанная мною функция работает довольно таки точно, не хуже, чем встроенные функции системы $Wolfram\ Mathematica$

Также для еще одного подтвержения правильности собственной реализации рассмотрим диаграмму разброса данных:

$$\text{ListPlot}[p\ [\![30;;130]\!] \rightarrow \text{Range}[\text{Length}[p\ [\![30;;130]\!]\ ]], \text{PlotStyle} \rightarrow \text{PointSize}[\text{Large}]]$$

На данном графике видно, что с повышением числа k найденное решение на k итерации становится все ближе и ближе к искомому решению. Сами решения «кучкуются» вокруг искомого решения.


1.1221
1.1220
1.1219
1.1218
1.1217
1.1216
1.5026
1.5027
1.5028
1.5029
1.5030
1.5031
1.5032
28
29
34
40
35
45
39
46
30
33
41
36
44
42
38
43
32
31
37
25