ISSN: 2181-3337 SCIENCE AND INNOVATION
INTERNATIONAL SCIENTIFIC JOURNAL
2022 № 3

РЕШЕНИЕ СИСТЕМ НЕЛИНЕЙНЫХ УРАВНЕНИЙ В MATLAB

М.Н.Урманов
НамМСИ Учитель
Н.Д.Нуритдинов
НамМСИ Учитель
А.Алиева
НамМСИ студент

https://doi.org/10.5281/zenodo.6762248

Аннотация. *Для Решение систем нелинейных уравнений в Matlabe. Для решения СНАУ, необходимо воспользоваться* ***итеративными методами****. Это методы, которые за определенное количество шагов получают решение с определенной точностью. Также очень важно при решении задать* ***достаточно близкое начальное приближение****, то есть такой набор переменных, которые близки к решению. Если решается система из 2 уравнений, то приближение находится с помощью построение графика двух функций.*

Ключевые слова*: начальное, приближение, итерация, нелинейный, Ньютона, функции, diff.*

SOLVING SYSTEMS OF NONLINEAR EQUATIONS IN MATLAB

Abstract. *For Solving systems of nonlinear equations in Matlab. To solve SNAU, iterative methods must be used. These are methods that, in a certain number of steps, obtain a solution with a certain accuracy. It is also very important when solving to set a sufficiently close initial approximation, that is, such a set of variables that are close to the solution. If a system of 2 equations is being solved, then the approximation is found by plotting two functions.*

Keywords: *initial, approximation, iteration, non-linear, Newtonian, functions, diff.*

ВВЕДЕНИЕ

Система представляет собой **набор нелинейных уравнений** (их может быть два или более), для которых иногда возможно найти решение, которое будет подходить ко всем уравнениям в системе.

$$\begin{cases} F_1(x) = F_1(x_1, x_2, \ldots, x_n) = 0, \\ F_2(x) = F_2(x_1, x_2, \ldots, x_n) = 0, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ F_n(x) = F_n(x_1, x_2, \ldots, x_n) = 0. \end{cases}$$

В стандартном виде, количество неизвестных переменных равно количеству уравнений в системе. Необходимо найти набор неизвестных переменных, которые при подставлении в уравнения будут приближать значение уравнения к 0. Иногда таких наборов может быть несколько, даже бесконечно много, а иногда решений не существует.

Чтобы решить СНАУ, необходимо воспользоваться **итеративными методами**. Это методы, которые за определенное количество шагов получают решение с определенной точностью. Также очень важно при решении задать **достаточно близкое начальное приближение**, то есть такой набор переменных, которые близки к решению. Если решается система из 2 уравнений, то приближение находится с помощью построение графика двух функций.

Далее, мы рассмотрим **стандартный оператор Matlab** для решения систем нелинейных алгебраических уравнений, а также напишем **метод простых итераций** и **метод Ньютона**.

Оператор Matlab для решения СНАУ

В среде Matlab существует оператор **fsolve**, который позволяет решить систему нелинейных уравнений. Сразу рассмотрим задачу, которую, забегая вперед, решим и другими методами для проверки.

Решить систему нелинейных уравнений с точность 10^{-2}:

cos(x-1) + y = 0.5

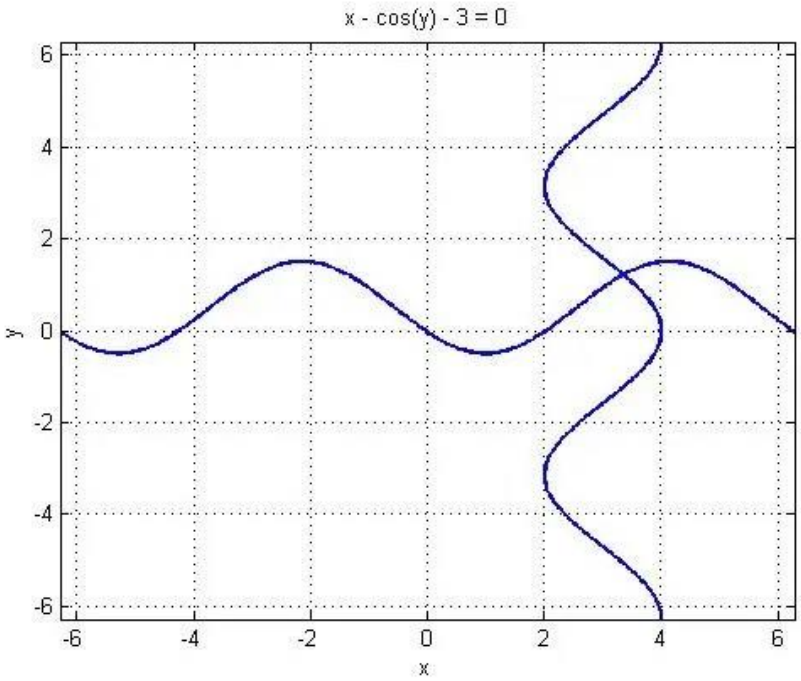
x-cos(y) = 3

МАТЕРИАЛЫ И МЕТОДЫ

Нам дана система из 2 нелинейных уравнений и сначала лучше всего построить график. Воспользуемся командой **ezplot** в Matlab, только не забудем преобразовать уравнения к стандартному виду, где правая часть равна 0:

```
y1 = ezplot('cos(x-1) + y - 0.5');
set(y1,'Color','b','LineWidth',2);
hold on;
y2 = ezplot('x - cos(y) - 3');
set(y2,'Color','b','LineWidth',2);
grid on;
```

Функция ezplot строит график, принимая символьную запись уравнения, а для задания цвета и толщины линии воспользуемся функцией set. Посмотрим на вывод:



Как видно из графика, есть одно пересечение функций — то есть одно единственное решение данной системы нелинейных уравнений. И, как было сказано, по графику найдем приближение. Возьмем его как (3.0, 1.0). Теперь найдем решение с его помощью:

Создадим функцию m-файлом fun.m и поместим туда следующий код:

```
function f = fun( x )
   f(1)= cos(x(1)-1) + x(2) - 0.5;
   f(2)= x(1) - cos(x(2)) - 3;
end
```

Заметьте, что эта функция принимает вектор приближений и возвращает вектор значений функции. То есть, вместо x здесь x(1), а вместо y — x(2). Это необходимо, потому что fsolve работает с векторами, а не с отдельными переменными.

И наконец, допишем функцию fsolve к коду построения графика таким образом:

```
%построение графика
y1 = ezplot('cos(x-1) + y - 0.5');
set(y1,'Color','b','LineWidth',2);
hold on;
y2 = ezplot('x - cos(y) - 3');
set(y2,'Color','b','LineWidth',2);
grid on;
%расчет решения
[xr, fr, ex] = fsolve(@fun,[3.0, 1.0],optimset('TolX',1.0e-2))
hold on;
plot(xr(1), xr(2), '*r');
```

Таким образом у нас образуется два m-файла. Первый строит график и вызывает функцию fsolve, а второй необходим для расчета самих значений функций

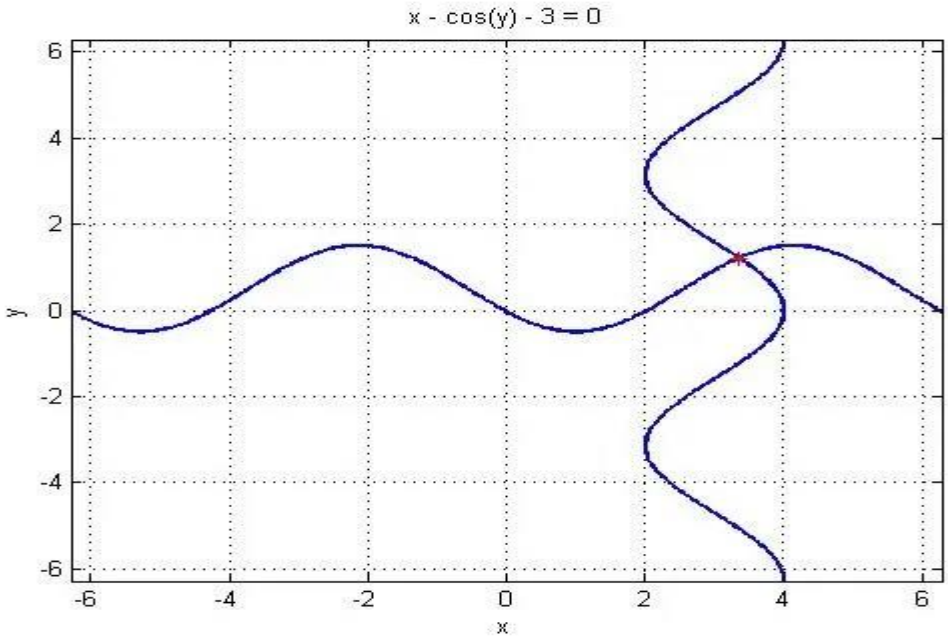
И в конце, приведем результаты:

xr (это вектор решений) = 3.3559 1.2069

fr (это значения функций при таких xr, они должны быть близки к 0) = 1.0e-09 *
0.5420 0.6829

ex (параметр сходимости, если он равен 1, то все сошлось) = 1

И, как же без графика с ответом:



`

Метод простых итераций в Matlab для решения СНАУ

РЕЗУЛЬТАТЫ

Теперь переходим к методам, которые запрограммируем сами. Первый из них — метод простых итераций. Он заключается в том, что итеративно приближается к решению, конечно же, с заданной точностью. Алгоритм метода достаточно прост:

1. Если возможно, строим график.
2. Из каждого уравнения выражаем неизвестную переменную след. образом: из 1 уравнения выражаем x1, из второго — x2, и т.д.
3. Выбираем начальное приближение X0, например (3.0 1.0)
4. Рассчитываем значение x1, x2,.. xn, которые получили на шаге 2, подставив значения из приближения X0.
5. Проверяем условие сходимости, (X-X0) должно быть меньше точности
6. Если 5 пункт не выполнился, то повторяем 4 пункт.

И перейдем к практике, тут станет все понятнее.

Решить систему нелинейных уравнений методом простых итераций с точность 10^{-2}:

cos(x-1) + y = 0.5

x-cos(y) = 3

График мы уже строили в предыдущем пункте, поэтому переходим к преобразованию. Увидим, что x из первого уравнения выразить сложно, поэтому поменяем местами уравнения, это не повлияет на решение:

x-cos(y) = 3

cos(x-1) + y = 0.5

Далее приведем код в Matlab:

```
x0 = 3.0; %точки полученные графически
y0 = 1.0; %возьмем их за начальное приближение
x = cos(y0) + 3;
y = 0.5 - cos(x - 1);

e = 0.01; %точность
k = 0; %счетчик итераций
```

В этой части мы выразили x1 и x2 (у нас это ‘x’ и ‘y’) и задали точность.

```
while or((abs(x - x0 ) > e), (abs(y - y0) > e)) %пока точность не будет достигнута
x0 = x;
y0 = y;
x = cos(y0) + 3;
y = 0.5 - cos(x - 1);
k = k + 1;
end;
```

ОБСУЖДЕНИЕ

В этой части в цикле выполняются пункты 4-6. То есть итеративно меняются значения x и y, пока отличия от предыдущего значения не станет меньше заданной точности.

Далее, выведем:

```
%вывод
```

k %количество итераций
x %решения
y
k = 10
x = 3.3587
y = 1.2088

Как видно, результаты немного отличаются от предыдущего пункта. Это связано с заданной точностью, можете попробовать поменять точность и увидите, что результаты станут такими же, как и при решении стандартным методом Matlab.

Метод Ньютона в Matlab для решения СНАУ

Решение систем нелинейных уравнений в Matlab методом Ньютона является более эффективным, чем использование метода простых итераций. Сразу же представим алгоритм, а затем перейдем к реализации.

1. Если возможно, строим график.
2. Выбираем начальное приближение X0, например (3.0 1.0)
3. Рассчитываем матрицу Якоби w, это матрица частных производных каждого уравнения, считаем ее определитель для X0.
4. Находим вектор приращений, который рассчитывается как $dx = -w^{-1} * f(X0)$
5. Находим вектор решения X = X0 + dx
6. Проверяем условие сходимости, (X-X0) должно быть меньше точности

Далее, решим тот же пример, что и в предыдущих пунктах. Его график мы уже строили и начальное приближение останется таким же.

Решить систему нелинейных уравнений методом Ньютона с точность 10^{-2}:
cos(x-1) + y = 0.5
x-cos(y) = 3

Перейдем к коду:

```
x0 = [3; 1];
syms xs ys; %создаем символьные переменные, чтобы вычислить производную
df1xs = diff('cos(xs-1) + ys - 0.5', xs); %производная 1 функции по x
df1ys = diff('cos(xs-1) + ys - 0.5', ys); %производная 1 функции по y
df2xs = diff('xs - cos(ys) - 3', xs); %производная 2 функции по x
df2ys = diff('xs - cos(ys) - 3', ys); %производная 2 функции по y
```

Сначала зададим начальное приближение. Затем необходимо просчитать матрицу Якоби, то есть частные производные по всем переменным. Воспользуемся символьным дифференцированием в Matlab, а именно командой diff с использованием символьных переменных.

Далее, сделаем первую итерацию метода, чтобы получить вектор выходных значений X, а потом уже сравнивать его с приближением в цикле.

```
f0=[cos(x0(1)-1) + x0(2) - 0.5; x0(1) - cos(x0(2)) - 3]; %помещаем функции в вектор столбец
df1x = double(subs(subs(df1xs, xs, x0(1)), ys, x0(2))); %определяем значение производных подставляя значение из начального приближения
df1y = double(subs(subs(df1ys, xs, x0(1)), ys, x0(2)));
df2x = double(subs(subs(df2xs, xs, x0(1)), ys, x0(2)));
```

```
df2y = double(subs(subs(df2ys, xs, x0(1)), ys, x0(2)));

w = [df1x df1y; df2x df2y];  %матрица Якоби
det(w); % определитель, не должен быть равен 0
x = x0 - inv(w) * f0; %решение после первой итерации

e = 0.01; %точность
k = 1;;
```

В этой части кода выполняем первую итерацию, чтобы получить вектор решения и сравнивать его с вектором начального приближения. Отметим, чтобы посчитать значение символьной функции в Matlab, необходимо воспользоваться функцией subs. Эта функция заменяет переменную на числовое значение. Затем функция double рассчитает это числовое значение.

Все действия, которые были выполнены для расчета производных, на самом деле можно было не производить, а сразу же задать производные. Именно так мы и поступим в цикле.

```
while or((abs(x(1) - x0(1)) > e), (abs(x(2) - x0(2)) > e)) %пока точность не будет достигнута
   x0(1) = x(1);
   x0(2) = x(2);
   f0=[cos(x0(1)-1) + x0(2) - 0.5; x0(1) - cos(x0(2)) - 3];
   w = [(-sin(x0(1) - 1)) 1; 1 sin(x0(2))]; %матрица Якоби
   det(w); % определитель, не должен быть равен 0
   x = x0 - inv(w) * f0; % новое решение
   k = k + 1;
   if ( k > 100)
     break;
   end;
end;
```

ВЫВОДЫ

В этой части кода выполняется цикл по расчету решения с заданной точностью. **Еще раз отметим, что если в первой итерации до цикла были использованы функции diff, double и subs для вычисления производной в Matlab, то в самом цикле матрица якоби была явно задана этими частными производными.** Это сделано, чтобы показать возможности среды Matlab.

Вывод:

```
x = 3.3559
1.2069
k = 3
```

За 3 итерации достигнут правильный результат. Также важно сказать, что иногда такие методы могут зацикливаться и не закончить расчеты. Чтобы такого не было, мы прописали проверку на количество итераций и запретили выполнение более 100 итераций.

Литература:

1. Arens V.Zh. Physical and chemical geotechnology. –M .: MGGU, 2001. –656 p.

2. 2. Konovalov A.N. The method of fictitious domains in problems of filtration of a two-phase incompressible fluid taking into account capillary forces // Numerical methods of continuum mechanics. T.3. –Novosibirsk: Computing Center of the Siberian Branch of the USSR Academy of Sciences, 1972. –No5. -S. 52-67.
3. Samarskiy A.A. Introduction to the theory of difference schemes. –M .: Nauka, 1971. – 552 p.