МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по практической работе №3

по дисциплине «Вычислительная математика»

ТЕМА: РЕШЕНИЕ НЕЛИНЕЙНЫХ УРАВНЕНИЙ И СИСТЕМ

Студент гр. 0303 _____ Калмак Д.А.

Преподаватель _____ Сучков А.И.

Санкт-Петербург

2021

Цель работы.

Сформировать представление о методах решения нелинейных уравнений и систем нелинейных уравнений, выработать умение составлять и применять алгоритмы для решения уравнений и систем уравнений, привить навык использования программных средств для решения нелинейных уравнений и систем нелинейных уравнений.

Основные теоретические положения.

Решение нелинейных уравнений

Численное решение нелинейных (алгебраических или трансцендентных) уравнений вида $f(x) = 0$ заключается в нахождении значений x, удовлетворяющих (с заданной точностью) данному уравнению и состоит из следующих основных этапов:

1. Отделение (изоляция, локализация) корней уравнения.
2. Уточнение с помощью некоторого вычислительного алгоритма конкретного выделенного корня с заданной точностью.

Целью первого этапа является нахождение отрезков из области определения функции $f(x)$, внутри которых содержится только один корень решаемого уравнения. Иногда ограничиваются рассмотрением лишь какой-нибудь части области определения, вызывающей по тем или иным соображениям интерес. Для реализации данного этапа используются графические или аналитические способы.

При аналитическом способе отделения корней полезна следующая теорема.

Теорема. Непрерывная строго монотонная функция $f(x)$ имеет и притом единственный нуль на отрезке $[a,b]$ тогда и только тогда, когда на его концах она принимает значения разных знаков.

Достаточным признаком монотонности функции $f(x)$ на отрезке $[a,b]$ является сохранение знака производной функции.

Графический способ отделения корней целесообразно использовать в том случае, когда имеется возможность построения графика функции $y = f(x)$.

Наличие графика исходной функции дает непосредственное представление о количестве и расположении нулей функции, что позволяет определить промежутки, внутри которых содержится только один корень. Если построение графика функции $y = f(x)$ вызывает затруднение, часто оказывается удобным преобразовать уравнение $f(x) = 0$ к эквивалентному виду $f1(x) = f2(x)$ и построить графики функций $y = f1(x)$ и $y = f2(x)$. Абсциссы точек пересечения этих графиков будут соответствовать значениям корней решаемого уравнения.

Так или иначе, при завершении первого этапа, должны быть определены промежутки, на каждом из которых содержится только один корень уравнения.

Для уточнения корня с требуемой точностью применяются различные итерационные методы, заключающиеся в построении числовой последовательности $x(k)$ $(k=0,1,2,...)$, сходящейся к искомому корню $x(*)$ уравнения $f(x) = 0$.

Решение систем нелинейных уравнений

Систему нелинейных уравнений с n неизвестными можно записать в виде

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) = 0 \\ f_2(x_1, x_2, \ldots, x_n) = 0 \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ f_n(x_1, x_2, \ldots, x_n) = 0 \end{cases}$$

или, более коротко, в векторной форме $F(X) = 0$, где $X=(x1,x2,...,xn)_T$ – вектор неизвестных, $F=(f1,f2,...,fn)_T$ – вектор-функция.

В редких случаях для решения такой системы удается применить метод последовательного исключения неизвестных и свести решение исходной задачи к решению одного нелинейного уравнения с одним неизвестным. Значения других неизвестных величин находятся соответствующей подстановкой в конкретные выражения. Однако в подавляющем большинстве случаев для решения систем нелинейных уравнений используются итерационные методы.

В дальнейшем предполагается, что ищется изолированное решение нелинейной системы.

Как и в случае одного нелинейного уравнения, локализация решения может осуществляться на основе специфической информации по конкретной решаемой задаче (например, по физическим соображениям), и с помощью методов математического анализа. При решении системы двух уравнений, достаточно часто удобным является графический способ, когда месторасположение корней определяется как точки пересечения кривых *f1(x1,x2) = 0, f2(x1, x2) = 0* на плоскости *(x1, x2).*

Постановка задачи.

Численно решить уравнение и систему уравнений методами Ньютона и простых итераций с заданной точностью ε. Значение ε варьируется от 0.1 до 0.000001.

Выполнение работы.

1. Решение уравнения $3.3 * x^2 - 3.1 * x - 12.5 = 9.1 * cos(2.3 * x + 6.6)$.

С помощью графика были отделены корни. Они расположены на двух отрезках: [-2, -1] и [2, 3].

Для каждого корня методом Ньютона проведены вычисления по программе для разных ε. Результаты представлены в табл. 1.

Таблица 1 – Метод Ньютона для разных ε

Значение ε	Значение x1	Значение x2	Число итераций k
0.1	2.922385	-1.086965	3
0.01	2.922385	-1.086965	3
0.001	2.922480	-1.086965	4/3
0.0001	2.922480	-1.086981	4
0.00001	2.922478	-1.086981	5/4
0.000001	2.922478	-1.086981	5/4

Число итераций увеличивается с уменьшением значения точности, также число итераций зависит от изначального отрезка, на котором рассматривается корень.

Для наименьшего корня результат работы программы на каждой итерации представлен в табл. 2. С каждой итерацией уменьшается изменение значения x по модулю.

Таблица 2 – Метод Ньютона для наименьшего корня при ε = 0.000001

Значение k	Значение x(k)	Значение f(x(k))	Значение f'(x(k))	Значение –f(x(k))/f'(x(k))
0	-1.500000	8.674678	-13.487741	0.643153
1	-0.856847	-6.665276	-29.536280	-0.225664
2	-1.082511	-0.122681	-27.544999	-0.004454
3	-1.086965	-0.000437	-27.455009	-0.000016

Метод Ньютона требует больше итераций при меньшем значении ε и менее точным отрезком. С меньшим значением ε значения x точнее. Теоретические и экспериментальные данные совпадают.

Для каждого корня методом простых итераций проведены вычисления по программе для разных ε. Результаты представлены в табл. 3.

Таблица 3 – Метод простых итераций для разных ε

Значение ε	Значение x1	Значение x2	Число итераций k
0.1	2.922385	-1.086965	4
0.01	2.922385	-1.086965	4
0.001	2.922480	-1.086965	5/4
0.0001	2.922480	-1.086981	5
0.00001	2.922478	-1.086981	6/5
0.000001	2.922478	-1.086981	6/5

Уменьшение значения точности приводит к увеличению количества итераций. Также число итераций зависит от начального отрезка.

Для наименьшего корня результат работы программы на каждой итерации представлен в табл. 4. С каждой итерацией также уменьшается изменение значения x по модулю.

Таблица 4 – Метод простых итераций для наименьшего корня при ε = 0.000001

Значение k	Значение x(k)	Значение φ(x(k))
0	-1.500000	-0.856847
1	-0.856847	-1.082511
2	-1.082511	-1.086965
3	-1.086965	-1.086981
4	-1.086981	-1.086981

Метод простых итераций имеет большее число итераций, если значение точности вычислений меньше. При этом метод имеет большее число итераций, чем метод Ньютона. С меньшим значением ε значения x точнее. Теоретические и экспериментальные данные совпадают.

2. Решение системы уравнений: $\begin{cases} 4x - 3e^{0.8y} = 5y - 8e^{-0.5x} + 5 \\ 5x^4 + 7y^4 = 972 - 397x - 302y \end{cases}$

С помощью графика были отделены корни. Приближенные значения корней: (2; 0.5) и (-4; 3).

Для каждого корня методом Ньютона проведены вычисления по программе для разных ε. Результаты представлены в табл. 5.

Таблица 5 – Метод Ньютона для системы уравнений для разных ε

Значение ε	Значение $\vec{r}1=(x1,y1)$	Значение $\vec{r}2=(x2,y2)$	Число итераций k
0.1	1.975160, 0.369642	-4.077828, 2.724549	2
0.01	1.975160, 0.369642	-4.077812, 2.724330	2/3
0.001	1.975160, 0.369641	-4.077812, 2.724330	3/3
0.0001	1.975160, 0.369641	-4.077812, 2.724330	3/4
0.00001	1.975160, 0.369641	-4.077812, 2.724330	3/4
0.000001	1.975160, 0.369641	-4.077812, 2.724330	3/4

Метод Ньютона для системы уравнения проходит большее число итераций при более маленьком значении точности. В зависимости от приближенной точки число итераций тоже отличается.

Для первого корня результат работы программы на каждой итерации представлен в табл. 6. С каждой итерацией уменьшается изменение значения x и y по модулю.

Таблица 6 – Метод Ньютона для корня при ε = 0.000001

Значение k	Значение $\vec{r}(k)$	Значение f1($\vec{r}(k)$)	Значение f2($\vec{r}(k)$)	Значение $-J^{-1}(\vec{r}(k))\cdot F(\vec{r}(k))$
0	2.000000, 0.500000	-1.032439	53.437500	-0.025777; -0.127921
1	1.974223, 0.372079	-0.022411	0.223438	0.000937; -0.002437
2	1.975160, 0.369642	-0.000007	0.000137	0.000000; -0.000001

Для каждого корня методом простых итераций проведены вычисления по программе для разных ε. Результаты представлены в табл. 7.

Реализована функция lambda(), которая вычисляет мажорирующую матрицу Λ. $\Lambda = [a\ b;\ g\ d]$, где коэффициенты находятся с помощью решения системы, состоящей из уравнений:

$$1 + a * \frac{df1(x0, y0)}{dx} + b * \frac{df2(x0, y0)}{dx} = 0$$

$$a * \frac{df1(x0, y0)}{dy} + b * \frac{df2(x0, y0)}{dy} = 0$$

$$g * \frac{df1(x0, y0)}{dx} + d * \frac{df2(x0, y0)}{dx} = 0$$

$$1 + g * \frac{df1(x0, y0)}{dy} + d * \frac{df2(x0, y0)}{dy} = 0$$

Таблица 7 – Метод простых итераций для системы уравнений для разных ε

Значение ε	Значение $\vec{r}$1=(x1,y1)	Значение $\vec{r}$2=(x2,y2)	Число итераций k
0.1	1.975111, 0.369728	-4.077756, 2.728458	2
0.01	1.975111, 0.369728	-4.077848, 2.725040	2/3

Продолжение таблицы 7

0.001	1.975158, 0.369644	-4.077815, 2.724452	3/4
0.0001	1.975158, 0.369644	-4.077812, 2.724334	3/6
0.00001	1.975160, 0.369641	-4.077812, 2.724331	4/7
0.000001	1.975160, 0.369641	-4.077812, 2.724330	5/8

Уменьшении значения ε приводит к увеличению количества итераций. Также число итераций зависит от начальной приближенной точки.

Для первого корня результат работы программы на каждой итерации представлен в табл. 8. С каждой итерацией также уменьшается изменение значения x и y по модулю.

Таблица 8 – Метод простых итераций для корня при ε = 0.000001

Значение k	Значение $\vec{r}(k)$	Значение $\Phi(\vec{r}(k))$
0	2.000000, 0.500000	1.974223, 0.372079
1	1.974223, 0.372079	1.975111, 0.369728
2	1.975111, 0.369728	1.975158, 0.369644
3	1.975158, 0.369644	1.975160, 0.369641
4	1.975160, 0.369641	1.975160, 0.369641

Разработанный код см. в Приложении А.

Выводы.

Таким образом, было численно решено уравнение методами Ньютона и простых итераций, а также решена система уравнений методами Ньютона и простых итераций при разных значениях ε. Метод Ньютона занимает меньшее число итераций. Оба метода имеют большее число итераций при меньшем значении ε, а также число зависит от начального приближения.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл Newtoneq.m

```
function Newtoneq()
  ezplot('3.3*x^2-3.1*x-12.5-9.1*cos(2.3*x+6.6)')
  grid on
  title("График для метода Ньютона для уравнения")
  e = 1e-6;

  d = [2, 3];
  [x1 n] = newton(d,e);
  printf('Корень уравнения x1 = %f, \n', x1);
  printf('Количество итераций равно %d. \n',n);

  d = [-2, -1];
  [x2 n] = newton(d,e);
  printf('Корень уравнения x2 = %f, \n', x2);
  printf('Количество итераций равно %d. \n',n);
  hold off
endfunction

function [xk k] = newton(range,e)
  a = range(1);
  b = range(2);

  x0 = (a+b)/2;
  y0 = resh(a);
  k = 0;
  while abs(y0) > e
    [f,df] = resh(x0);
    xk = x0 - f/df;
    [f, df] = resh(xk);
    x0 = xk;
    y0 = f;
    k = k + 1;
  endwhile
endfunction

function [f,df] = resh(x)
  f = 3.3*x^2-3.1*x-12.5-9.1*cos(2.3*x+6.6);
  df = (20839*sin((229*x+660)/100))/1000+(33*x)/5-31/10;
endfunction
```

Файл Iteration.m

```
function Iteration()
  ezplot('3.3*x^2-3.1*x-12.5-9.1*cos(2.3*x+6.6)')
  grid on
  title("График для метода простых итераций для уравнения")
  e = 1e-6;

  d = [2, 3];
  [x1 n] = siter(d,e);
  printf('Корень уравнения x1 = %f, \n', x1);
```

```
  printf('Количество итераций равно %d. \n',n);

  d = [-2, -1];
  [x2 n] = siter(d,e);
  printf('Корень уравнения x2 = %f, \n', x2);
  printf('Количество итераций равно %d. \n',n);
  hold off
endfunction

function [xk,k] = siter(range,e)
  k = 0;
  a = range(1);
  b = range(2);
  x0 = (a+b)/2;
  xk = x0;
  while abs(resh(xk)) > e
      xk = x0;
      x0 = phi(xk);
      k = k+1;
  end
endfunction

function x = phi(xk)
   df = reshdf(xk);
   g = -1/(df);
   x = xk + g*resh(xk);
endfunction

function f = resh(x)
  f = 3.3*x.^2-3.1*x-12.5-9.1*cos(2.3*x+6.6);
endfunction

function df = reshdf(x)
  df = (20839*sin((229*x+660)/100))/1000+(33*x)/5-31/10;
endfunction
```

Файл Newtonsys.m

```
function Newtonsys()
  ezplot('4*x-3*e^(0.8*y) - 5*y +8*e^(-0.5*x)-5')
  hold on
  grid on
  ezplot('5*x^4+7*y^4 - 972+397*x+302*y')
  title("График для метода Ньютона для системы уравнений")
  e = 1e-6;

  x0 = [2; 0.5];
  [x1 n] = newts(x0,e);
  printf('Корень уравнения x1 = %f y1 = %f, \n', x1);
  printf('Количество итераций равно %d. \n',n);

  x0 = [-4; 3];
  [x2 n] = newts(x0,e);
  printf('Корень уравнения x2 = %f y2 = %f, \n', x2);
  printf('Количество итераций равно %d. \n',n);
  hold off
endfunction
```

```
function [x,k] = newts(x0,e)
  f1 = @(x,y)  4*x-3*exp(0.8*y)-5*y+8*exp(-0.5*x)-5;
  f2 = @(x,y)  5*x^4+7*y^4-972+397*x+302*y;
  f0 = [f1(x0(1),x0(2)); f2(x0(1),x0(2))];

  df1x = @(x) 4-4*exp(-(x/2));
  df1y = @(y) (-(12*exp((4*y)/5))/5)-5;
  df2x = @(x) 20*x^3+397;
  df2y = @(y) 28*y^3+302;

  J = [df1x(x0(1)) df1y(x0(2)); df2x(x0(1)) df2y(x0(2))];
  x = x0 - inv(J) * f0;
  k = 1;

  while or((abs(x(1) - x0(1)) > e), (abs(x(2) - x0(2)) > e))
      x0(1) = x(1);
      x0(2) = x(2);
      f0 = [f1(x0(1),x0(2)); f2(x0(1),x0(2))];

      J = [df1x(x0(1)) df1y(x0(2)); df2x(x0(1)) df2y(x0(2))];
      x = x0 - inv(J) * f0;
      k = k + 1;
  end;
endfunction
```

Файл Iterationsys.m

```
function Iterationsys()
  ezplot('4*x-3*e^(0.8*y) - 5*y +8*e^(-0.5*x)-5')
  hold on
  grid on
  ezplot('5*x^4+7*y^4 - 972+397*x+302*y')
  title("График для метода простых итераций для системы уравнений")
  e = 1e-6;

  x0 = [2; 0.5];
  [x1 n] = siters(x0,e);
  printf('Корень уравнения x1 = %f y1 = %f, \n', x1);
  printf('Количество итераций равно %d. \n',n);

  x0 = [-4; 3];
  [x2 n] = siters(x0,e);
  printf('Корень уравнения x2 = %f y2 = %f, \n', x2);
  printf('Количество итераций равно %d. \n',n);
  hold off
endfunction

function [x,k] = siters(x0,e)
  L = lambda(x0);
  x = Phi(x0, L);
  k = 1;

  while or((abs(x(1) - x0(1)) > e), (abs(x(2) - x0(2)) > e))
      x0(1) = x(1);
      x0(2) = x(2);

      x = Phi(x0, L);
      k = k + 1;
```

```
  end;
endfunction

function l = lambda(x0)
  x = x0(1);
  y = x0(2);
  df1x = 4-4*exp(-(x/2));
  df1y = (-(12*exp((4*y)/5))/5)-5;
  df2x = 20*x^3+397;
  df2y = 28*y^3+302;

  a = (df2y)/(df1y*df2x - df2y*df1x);
  b = -a*df1y/df2y;
  g = (df2x)/(df2y*df1x - df2x*df1y);
  d = -g*df1x/df2x;

  l = [a b; g d];
endfunction

function x = Phi(x0, L)
   f1 = @(x,y)  4*x-3*exp(0.8*y)-5*y+8*exp(-0.5*x)-5;
   f2 = @(x,y)  5*x^4+7*y^4-972+397*x+302*y;
   f0 = [f1(x0(1),x0(2)); f2(x0(1),x0(2))];
   x = x0 + L * f0;
endfunction
```