**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА**

**Лабораторна робота 1.2-beta**

Зміст:

1. Постановка задачі.
2. Обґрунтування методів.
3. Результати.
4. Код програми.
5. Висновок.

1. Постановка задачі.

 Знайти розв’язок системи нелінійних рівнянь

$\left\{\begin{array}{c}\cos(\left(x^{2}+y^{2}\right))-x+y=a\\\frac{x+y-2}{k}+\frac{(x-y)^{2}}{k-0.1}=1\end{array}\right.$,

використовуючи наступні методи:

1. метод простих ітерацій;
2. метод Ньютона.

Розв’язати з точністю до 0,001 і визначити кількість ітерацій.

2. Обґрунтування методів.

Пусть требуется решить систему уравнений

 (2.1)

где— заданные, вообще говоря, нелинейные (среди них могут быть и линейные) вещественнозначные функции *п* вещественных переменных Обозначив

, , 

данную систему (2.1) можно записать одним уравнением

 (2.1а)

относительно векторной функции *F* векторного аргумента х. Таким образом, исходную задачу можно рассматривать как зада­чу о нулях нелинейного отображения В этой постановке она является прямым обобщением основной задачи предыдущей главы — задачи построения методов нахождения нулей одномерных нелинейных отображений. Фактически это та же задача, только в пространствах большей размерности. Поэтому можно как заново строить методы ее решения на основе разработанных выше подходов, так и осуществлять формальный перенос выведенных для скалярного случая расчетных формул. В любом случае следует позаботиться о правомочности тех или иных операций над векторными переменными и векторными функциями, а также о сходимости получаемых таким способом итерационных процессов. Часто теоремы сходимости для этих процессов являются тривиальными обобщениями соответствующих результатов, полученных для методов решения скалярных уравнений. Однако не все результаты и не все методы можно перенести со случая *п* = 1 на случай *п* ≥2. Например, здесь уже не будут работать методы дихотомии, поскольку множество векторов не упорядочено. В то же время, переход от *n* = 1 до *n≥*2 вносит в задачу нахождения нулей нелинейного отображения свою специфику, учет которой приводит к новым методам и к различным модификациям уже имеющихся. В частности, большая вариативность методов решения нелинейных систем связана с разнообразием способов, которыми можно решать линейные алгебраические задачи, возникающие при пошаговой линеаризации данной нелинейной вектор-функции *F(x).*

**МЕТОД НЬЮТОНА**

Пусть () — некоторая последовательность невырожденных вещественных n x n-матриц. Тогда, очевидно, последовательность задач

,  *k = 0,1,2,...*

имеет те же решения, что и исходное уравнение (2.1а), и для приближенного нахождения этих решений можно формально записать итерационный процесс

, *k = 0,1,2,...* (3.1.1)

имеющий вид метода простых итераций (4.2.1b) при . В случае  - это действительно МПИ с линейной сходимостью последовательности () Если же  различны при разных *k*, то формула (3.1.1) определяет большое семейство итерационных методов с матричными параметрами. Рассмотрим некоторые из методов этого семейства.

Положим , где

— матрица Якоби вектор-функции *F(x).* Подставив это  в (3.1.1), получаем явную формулу метода Ньютона

, (3.1.2)

обобщающего на многомерный случай скалярный метод Ньютона (5.14). Эту формулу, требующую обращения матриц на каждой итерации, можно переписать в неявном виде:

. (3.1.3)

Применение (3.1.3) предполагает при каждом *k = 0,1,2,...* решение линейной алгебраической системы

относительно векторной поправки , а затем прибавление этой поправки к текущему приближению для получения следующего:

.

К решению таких линейных систем можно привлекать самые разные методы как прямые, так и итерационные в зависимости от размерности n решаемой задачи и специфики матриц Якоби  (например, можно учитывать их симметричность, разреженность и т.п.).

Сравнивая (3.1.3) с формальным разложением *F(x)* в ряд Тейлора

,

видим, что последовательность () в методе Ньютона получается в результате подмены при каждом *k=0,1,2,...* нелинейного уравнения *F(x) = 0* или, что то же (при достаточной гладкости *F(x)*), уравнения

линейным уравнением

т. е. с пошаговой линеаризацией. Как следствие этого факта, можно рассчитывать, что при достаточной гладкости *F(x)* и достаточно хорошем начальном приближении  сходимость порождаемой методом Ньютона последовательности () к решению  будет квадратичной и в многомерном случае. Имеется ряд теорем, устанавливающих это при тех или иных предположениях. В частности, одна из таких теорем приводится ниже.

Новым, по сравнению со скалярным случаем, фактором, осложняющим применение метода Ньютона к решению n-мерных систем, является необходимость решения n-мерных линейных задач на каждой итерации (обращения матриц в (3.1.2) или решения СЛАУ в (3.1.3)), вычислительные затраты на которые растут с ростом n, вообще говоря, непропорционально быстро. Уменьшение таких затрат — одно из направлений модификации метода Ньютона.

***Доказательство для одного уравнения, для системы - аналогичное:*** Пусть дана [функция](http://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29) [действительного переменного](http://ru.wikipedia.org/wiki/%D0%92%D0%B5%D1%89%D0%B5%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE) дважды [непрерывно дифференцируемая](http://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%BF%D1%80%D0%B5%D1%80%D1%8B%D0%B2%D0%BD%D0%B0%D1%8F_%D0%B4%D0%B8%D1%84%D1%84%D0%B5%D1%80%D0%B5%D0%BD%D1%86%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D0%BE%D1%81%D1%82%D1%8C) в своей [области определения](http://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BB%D0%B0%D1%81%D1%82%D1%8C_%D0%BE%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F), [производная](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B8%D0%B7%D0%B2%D0%BE%D0%B4%D0%BD%D0%B0%D1%8F) которой нигде не обращается в нуль:



И необходимо доказать, что функция осуществляет сжимающее отображение вблизи корня уравнения . В силу [непрерывной дифференцируемости](http://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%BF%D1%80%D0%B5%D1%80%D1%8B%D0%B2%D0%BD%D0%B0%D1%8F_%D0%B4%D0%B8%D1%84%D1%84%D0%B5%D1%80%D0%B5%D0%BD%D1%86%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D0%BE%D1%81%D1%82%D1%8C) функции и неравенства нулю её первой [производной](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B8%D0%B7%D0%B2%D0%BE%D0%B4%D0%BD%D0%B0%D1%8F) [непрерывна](http://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%BF%D1%80%D0%B5%D1%80%D1%8B%D0%B2%D0%BD%D0%B0%D1%8F_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F). Производная равна:



В условиях, наложенных на , она также непрерывна. Пусть  — искомый корень уравнения: , следовательно в его окрестности :



Тогда согласно [теореме Лагранжа](http://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D0%BC%D1%83%D0%BB%D0%B0_%D0%BA%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D1%8B%D1%85_%D0%BF%D1%80%D0%B8%D1%80%D0%B0%D1%89%D0%B5%D0%BD%D0%B8%D0%B9):



В силу того, что в этой же дельта окрестности выполняется:



Таким образом полученная функция в окрестности корня осуществляет [сжимающее отображение](http://ru.wikipedia.org/wiki/%D0%A1%D0%B6%D0%B8%D0%BC%D0%B0%D1%8E%D1%89%D0%B5%D0%B5_%D0%BE%D1%82%D0%BE%D0%B1%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5).

**МЕТОД ПРОСТЫХ ИТЕРАЦИЙ**

Пусть система (2.1) имеет вид (преобразована к виду):

 (4.2.1)

или иначе, в компактной записи,

, (4.2.1а)

где

.

Для этой ***задачи о неподвижной точке*** нелинейного отображения запишем формально рекуррентное равенство,

 (4.2.1b)

которое определяет ***метод простых итераций (МПИ)*** (или ***метод последовательных приближений)*** для задачи (4.2.1).

Если начать процесс построения последовательности () с некоторого вектора  и продолжить по формуле (4.2.1b), то при определенных условиях эта последовательность со скоростью геометрической прогрессии будет приближаться к вектору — неподвижной точке отображения *Ф(х).* А именно, справедлива следующая теорема.

**Теорема 4.1.**

*Пусть функция* Ф(х) *и замкнутое множество*  *таковы, что:*

*1)*  ;

2)  

*Тогда* Ф(х) *имеет в М единственную неподвижную точку* ; *последовательность (**), определяемая МПИ* (4.2.1b), *при любом*  *сходится к*  *и справедливы оценки*

 

**Теорема 4.2.**

*Пусть функция* Ф(х) *дифференцируема в замкнутом шаре* *причем* :

*. Тогда, если центр*  *и радиус r шара S таковы, что*  *, то справедливo заключение теоремы 4.1 с М=S.*

Если потребовать непрерывную дифференцируемость *Ф(х)*, то более просто перейти от теоремы 4.1 к теореме 4.2, применив следующее утверждение.

**Лемма 4.1.** *Пусть функция*  *непрерывна и дифференцируема на множестве*  *и пусть* . *Тогда* Ф(х) *удовлетворяет на множестве М условию Липшица*

.

Запись МПИ (4.2.1b) в развернутом виде, т.е. совокупность рекуррентных равенств



напоминает МПИ для СЛАУ, который укладывается в эту схему, если все функции — линейные. Учитывая, что в линейном случае, как правило, по сравнению с МПИ более эффективен метод Зейделя, здесь тоже может оказаться полезной модификация. А именно, вместо (4.2.1b) можно реализовать следующий ***метод итераций:***

 (4.2.2)

Заметим, что как и для линейных систем, отдельные уравнения в методе (4.2.2) неравноправны, т.е. перемена местами уравнений системы (4.2.1) может изменить в каких-то пределах число итераций и вообще ситуацию со сходимостью последовательности итераций. Чтобы применить метод простых итерации или его зейделеву модификацию (4.2.2) к исходной системе (2.1), нужно, как и в скалярном случае, сначала тем или иным способом привести ее к виду (4.2.1). Это можно сделать, например, умножив (2.1а) на некоторую неособенную n-x-n матрицу – **А** и прибавив к обеим частям уравнения -**A***F***(x)=0** вектор неизвестных **х**. Полученная система

**x = x-A***F***(x)**

эквивалентна данной и имеет вид задачи о неподвижной точке (4.2.1а). проблема теперь состоит лишь в подборе матричного параметра **А** такого, при котором вектор-функция **Ф(х):=х-А***F***(x)** обладала бы нужными свойствами.

**Підрахування констант:**

3. Результати.

4. Код програми.

***Блок-схема метода Ньютона для решения систем нелинейных уравнений***

****

## *Блок-схема алгоритма метода Гаусса без выбора главного элемента*



#include <cstdlib>

#include <iostream>

#include <cmath>

using namespace std;

const double eps = 0.001;

const double A = -0.3;

const double Z\_x0 = 0;

const double Z\_y0 = -1;

const double Ca = 0.0;

const double Ck = 0.4;

/// F(Z), x,y<=1

double func1(double x, double y){ return cos(x \* x + y \* y) - x + y - Ca;}

double func2(double x, double y)

{ return (x + y - 2) \* (x + y - 2) / Ck + (x - y) \* (x - y) / (Ck - 0.1) - 1;}

///J(Z)

double func11(double x, double y){ return -2\*x\*sin(x\*x+y\*y)-1;}

double func12(double x, double y){ return -2\*y\*sin(x\*x+y\*y)+1;}

double func21(double x, double y){ return 2/Ck\*(x+y-2) + 2/(Ck-0.1)\*(x-y);}

double func22(double x, double y){ return 2/Ck\*(x+y-2) - 2/(Ck-0.1)\*(x-y);}

/// вектор Z

double Z\_x = 0;

double Z\_y = -1;

int NIter = 0;// количество итераций

int N = 2;// количество уравнений в системе

double F[2];

double J[2][2];

double J1[2][2];

double J1F[2];

bool Newthon()

{

 ///F(Z):

 F[0] = func1(Z\_x, Z\_y);

 F[1] = func2(Z\_x, Z\_y);

 ///J(Z):

 J[0][0] = func11(Z\_x, Z\_y);

 J[0][1] = func12(Z\_x, Z\_y);

 J[1][0] = func21(Z\_x, Z\_y);

 J[1][1] = func22(Z\_x, Z\_y);

 ///J(Z)^(-1):

 double tmp = J[0][0] \* J[1][1] - J[0][1] \* J[1][0];

 J1[0][0] = J[1][1] / tmp;

 J1[0][1] = - J[0][1] / tmp;

 J1[1][0] = - J[1][0] / tmp;

 J1[1][1] = J[0][0] / tmp;

 /// [J(Z)]^(-1) \* F(Z):

 J1F[0] = J1[0][0] \* F[0] + J1[0][1] \* F[1];

 J1F[1] = J1[1][0] \* F[0] + J1[1][1] \* F[1];

 /// Z = Z - [J(Z)]^(-1) \* F(Z):

 Z\_x -= J1F[0];

 Z\_y -= J1F[1];

 /// if (||Z|| < eps) return true; else return false:

 if (sqrt(J1F[0] \* J1F[0] + J1F[1] \* J1F[1]) < eps) return true; else return false;

 return true;

}

bool Iter()

{

 ///F(Z):

 F[0] = func1(Z\_x, Z\_y);

 F[1] = func2(Z\_x, Z\_y);

 ///new X:

 Z\_x -= A \* F[0];

 Z\_y -= A \* F[1];

 if (sqrt((A \* F[0]) \* (A \* F[0]) + (A \* F[1]) \* (A \* F[1])) < eps) return true; else return false;

 return true;

}

int main(int argc, char \*argv[])

{

 NIter = 0; Z\_x = Z\_x0; Z\_y = Z\_y0;

 cout<<"Newthon:"<<endl;

 while(!Newthon())

 {

 NIter++;

 }

cout << NIter << ":" << Z\_x << "|" << Z\_y << ":" << endl;

 NIter = 0; Z\_x = Z\_x0; Z\_y = Z\_y0;

 cout<<"Iter:"<<endl;

 while(!Iter())

 {

 NIter++;

 }

cout << NIter << ":" << Z\_x << "|" << Z\_y << ":" << endl;

 system("PAUSE");

 return EXIT\_SUCCESS;

}

5. Висновок.

Методи зійшлися, отже зроблені вірно. Похибка для таких методів апріорі відома: 0.001 – адже заздалегідь ми задаємо точність підрахунків.