Федеральное государственное бюджетное образовательное учреждение высшего образования
"Уфимский государственный авиационный технический университет"

Кафедра Высокопроизводительных вычислительных технологий и систем

Дисциплина: Численные методы

Отчет по лабораторной работе № 4

Тема: «Решение нелинейных уравнений и их систем**»**

Группа ПМ-353	Фамилия И.О.	Подпись	Дата	Оценка
Студент	Шамаев И.Р.			
Принял				

Уфа 2021

Цель работы: получить навык численного решения нелинейных уравнений и систем таких уравнений.

Теоретический материал

Задача 1. Решение нелинейного уравнения разными методами

Метод бисекции (дихотомии).

Пусть мы нашли такие точки, x_0, x_1, что $f(x¿¿0)f(x_1)\le 0,¿$ т.е. на отрезке $[x_0, x_1]$ лежит не менее одного корня уравнения. Найдем середину отрезка $x_2=(x¿¿0+x_1)/2¿$ и вычислим $f(x¿¿2)¿$. Из двух половин отрезка выберем ту, для которой $f(x¿¿2)f(x¿¿гран)¿\le 0,¿$ ибо один из корней лежит на этой половине. Затем новый отрезок опять делим пополам и выберем ту половину, на концах которой функция имеет разные знаки, и т.д.

Если требуется найти корень с точностью ε, то продолжаем деление пополам до тех пор, пока длина отрезка не станет меньше 2ε.

Метод хорд.

Суть метода хорд состоит в разбиении отрезка $[x_0, x_1]$ (при условии $f(x¿¿0)f(x_1)\le 0¿$) на два отрезка с помощью хорды и выборе нового отрезка от точки пересечения хорды с осью абсцисс до неподвижной точки, на котором функция меняет знак и содержит решение, причём подвижная точка приближается к ε-окрестности решения.

Итерационная формула выглядит следующим образом

$$x_i = x_{i-1} - \frac{f(x_{i-1})}{f(x_{i-1}) - f(x_0)}(x_{i-1} - x_0).$$

Построение хорд продолжается до достижения необходимой точности решения ε.

Метод простых итераций.

Исходное уравнение $f(x)=0$ заменяется эквивалентным уравнением $x=\varphi(x)$. Далее выбирается некоторое нулевое приближение x_0 и дальнейшие приближения вычисляются по формуле

$$x_{n+1}=\varphi(x_n), n=0,1,2,\ldots.$$

Для сходимости, функцию $\varphi(x)$ берут в виде $\varphi(x)=x+\tau(x)f(x)$, причем функция $\tau(x)$ не меняет знака на том отрезке, где идет отыскание корня.

Данный метод сходится при надлежащем выборе начального приближения и если $|s'(x)|<1$, где x – корень уравнения.

Метод касательных (Ньютона).

Этот метод также методом касательных или методом линеаризации. Если x_nесть некоторое приближение к корню $\bar{x}$, а $f(x)$ имеет непрерывную производную, то уравнение $f(x)=0$ можно преобразовать следующим образом:

$$0=f(\bar{x})=f\left(x_n+(\bar{x}-x_n)\right)=f(x_n)+(\bar{x}-x_n)f'(\xi).$$

Приближенно заменяя $f'(\xi)$ на значение в известной точке x_n, получим следующий итерационный процесс:

$$x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)}.$$

Геометрически этот процесс означает замену на каждой итерации графика $y=f(x)$ касательной к нему.

Метод секущих.

В методе Ньютона требуется вычислять производную функции, что не всегда удобно. Можно заменить производную первой разделенной разностью, найденной по двум последним итерациям, т.е. заменить касательную секущей. Тогда получим

$$x_{n+1}=x_n-\frac{(x_n-x_{n-1})f(x_n)}{f(x_n)-f(x_{n-1})}.$$

Для начала процесса необходимо задать x_0 и x_1. Такие процессы, где для вычисления очередного приближения надо знать два предыдущих, называют двухшаговыми.

Задача 2. Решение системы двух нелинейных уравнений методом простых итераций

Систему нелинейных уравнений можно кратко записать в векторном виде

$$f(x)=0$$

Или более подробно в координатном виде

$$f_k(x_1,x_2,\dots,x_n)=0,1\le k\le n.$$

Нулевое приближение в случае двух переменных можно найти графически: построить на плоскости $(x¿¿1,x_2)¿$ кривые $f_1(x¿¿1,x_2)=0¿$ и $f_2(x¿¿1,x_2)=0¿$ и найти точки их пересечения.

Аналогично одномерному случаю метода простых итераций заменим нелинейную систему эквивалентной системой специального вида $x=\varphi(x)$. Выберем некоторое нулевое приближение и дальнейшие приближения найдем по формулам

$$x^{(s+1)}=\varphi(x^{(s)})$$

или

$$x_k^{(s+1)}=\varphi_k(x_1^{(s)},x_2^{(s)},\dots,x_n^{(s)}),1\le k\le n.$$

Если итерации сходятся, то они сходятся к решению уравнения (предполагается, что решение существует).

Обозначим за $M_{kl}=max\vee\frac{\partial\varphi_k}{\partial x_i}\vee$.Достаточным условием сходимости является $¿\vee M_{ki}\vee¿$<1.

Задача 3. Решение системы двух нелинейных уравнений методом Ньютона

Пусть известно некоторое приближение $x^{(s)}$ к корню $\bar{x}$. Как и для одной переменной, запишем исходную систему $f(x)=0$ в виде $f(x^{(s)}+\Delta x)=0$, где $\Delta x=\bar{x}-x^{(s)}$. Разлагая эти уравнения в ряды и ограничиваясь первыми дифференциалами, т.е. линеаризуя функцию, получим

$$\sum_{i=1}^{n}\frac{\partial f_k(x^{(s)})}{\partial x_i}\Delta x_i^{(s)}=-f_k(x^{(s)}),\ 1\le k\le n.$$

Это система уравнений, линейных относительно приращений $\Delta x_i^{(s)}$. Все коэффициенты этой матрицы выражаются через последнее приближение $x^{(s)}$. Решив эту систему (например, методом исключения), найдем новое приближение $x^{(s+1)}=x^{(s)}+\Delta x^{(s)}$.

Как и для одной переменной, метод Ньютона можно формально свести к методу последовательных приближений, положив $\varphi(x)=x-\left[\frac{\partial f}{\partial x}\right]^{-1}f(x)$, где $\left[\frac{\partial f}{\partial x}\right]^{-1}$ есть матрица, обратная матрице производных.

Критерий окончания ¿$\vee x^{(s)}-x^{(s+1)}\vee$¿$\le\varepsilon$.

Практическая часть

Задача 1.

Написать вычислительную программу на языке программирования C++ для решения нелинейного уравнения на указанном отрезке с заданной точностью методом

а) бисекции (дихотомии) (1 балл),
б) хорд (1 балл),
в) простых итераций (1 балл),
г) касательных (Ньютона) (1 балл),
д) секущих (1 балл).

Программа должна предусматривать возможность нахождения всех корней уравнения с заданной точностью.

1) С использованием написанной программы решить нелинейное уравнение согласно индивидуальному заданию.
2) Выполнить сравнительный анализ реализованных методов.

Описание:

Результат:

Задача 2.

1) Написать вычислительную программу на языке программирования C++ для решения системы двух нелинейных уравнений методом простых итераций с заданной точностью.
2) С использованием написанной программы найти численно минимум заданной функции двух переменных $F(x,y)$ в указанной области путем численного решения системы двух нелинейных уравнений, получающихся на основе необходимых условий экстремума.

$$F(x,y)=x^2+y^2+(x+y+1)\,tg(x+y),$$

где $x \times y$ искать в области $[-1/2,0]\times[-1/2,0]$ с заданной точностью 10^{-6}.

```
Точка минимума: (-0.0950122; 0.961125)
 F(-0.0950122; 0.961125) = -1.02573
 Количество итераций: 8
```

Рисунок 3. Пример выполнения программы 2

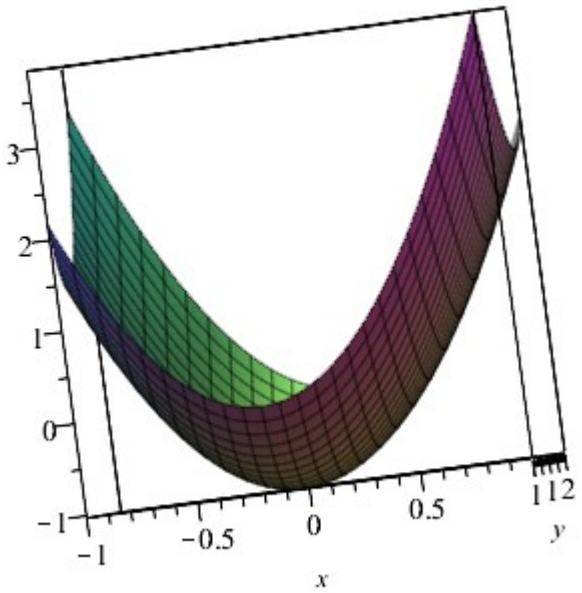

Рисунок 4. График функции по x

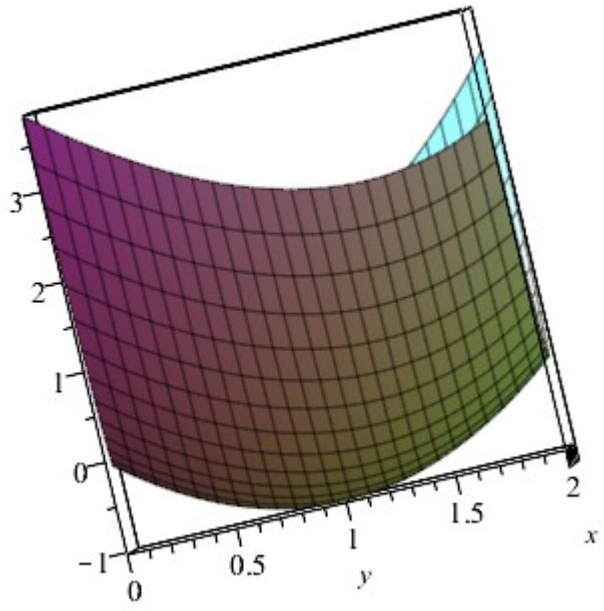

Рисунок 5. График функции по Y

То есть корень по X должен быть приближен к 0, а по Y к 1.

Задача 3

1) Написать вычислительную программу на языке программирования C++ для решения системы двух нелинейных уравнений методом Ньютона. При этом предусмотреть две возможности: а) точное задание всех производных, б) приближенное вычисление производных по точно заданным функциям с заданной точностью.
2) С использованием написанной программы решить задачу о поиске минимума функции двух переменных $F(x,y)$ сведением к системе двух нелинейных уравнений с использованием необходимого условия экстремума. Выполнить сравнительный анализ двух указанных в п.1) реализаций метода.

Рисунок 6. Пример выполнения программы 3

Вывод

В ходе лабораторной работы были получены навыки численного решения нелинейных уравнений и систем таких уравнений.

Список литературы

1. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. Численные методы.
2. Калиткин Н.Н. Численные методы.

Приложение В

Задание 2, 3

```
#include <iostream>
#include <cmath>
#include <math.h>
#include <vector>
#include<iomanip>
using namespace std;
const double eps = 10e-5;
//2 ЗАДАНИЕ
double F_x(double x, double y)
{
	return -cos(x) * cos(y) / 6.0;
}
double F_y(double x, double y) {

	return 1 + sin(x) * sin(y) * 0.5;
}

//3 ЗАДАНИЕ
double F1(double x, double y)
{

	return 6*x+cos(y)*cos(x);
}
double F2(double x, double y)
{

	return  2*y-2-sin(x)*sin(y);
}
double F1dx(double x, double y)
{
	return 6-cos(y)*sin(x);
}
double F1dy(double x, double y)
{
	return -cos(x)*sin(y);
}
double F2dx(double x, double y)
{

	return -sin(y)*cos(x);
}

double F2dy(double x, double y)
{
	return 2-sin(x)*cos(y);
}

double det(double x, double y)
{
	double det = F1dx(x, y) * F2dy(x, y) - F1dy(x, y) * F2dx(x, y);
	return det;
}


double FuncXY(double x, double y) {

	return 3*(x*x)+(y*y)-2*y+sin(x)*cos(y);
}

void SimpleIterationsMethod(double x0, double y0)
{
```

```
	int iteration = 0;
	double x_k1 = x0;
	double x_k2 = x_k1;
	double y_k1 = y0;
	double y_k2 = y_k1;
	double delta = 1;
	while (delta >= eps)
	{
		iteration++;
		x_k2 = x_k1;
		y_k2 = y_k1;
		x_k1 = F_x(x_k2, y_k2);
		y_k1 = F_y(x_k2, y_k2);
		delta = fmax(abs(x_k2 - x_k1), abs(y_k2 - y_k1));
		//    cout  << iteration << ": F(" << x_k1 << "; " << y_k1 << ") = "
<< FuncXY(x_k1, y_k1) << endl;
	}

	cout << "Точка минимума: (" << x_k1 << "; " << y_k1 << ")" << endl;
	cout << " F(" << x_k1 << "; " << y_k1 << ") = " << FuncXY(x_k1, y_k1) <<
endl;
	cout << " Количество итераций: " << iteration << endl;

}


void NewtonMethodExact(double x0, double y0)
{
	vector<double> root;
	int iteration = 0;
	double x_k = x0;
	double x_k1 = x_k;
	double y_k = y0;
	double y_k1 = y_k;
	double delta = 1;
	//Метод основан на матрице обратной к матрице Якоби
	//Значение Якобиана
	while (delta >= eps)
	{
		x_k1 = x_k;
		y_k1 = y_k;

		x_k = x_k1 - (1.0 / det(x_k1, y_k1)) * (F2dy(x_k1, y_k1) * F1(x_k1,
y_k1) - F1dy(x_k1, y_k1) * F2(x_k1, y_k1));
		y_k = y_k1 - (1.0 / det(x_k1, y_k1)) * (F1dx(x_k1, y_k1) * F2(x_k1,
y_k1) - F2dx(x_k1, y_k1) * F1(x_k1, y_k1));

		iteration++;
		delta = fmax(abs(x_k1 - x_k), abs(y_k1 - y_k));
		//cout << iteration << ": F(" << x_k << "; " << y_k << ") = " <<
FuncXY(x_k, y_k) << endl;

	}
	cout << "Точка минимума: (" << x_k1 << "; " << y_k1 << ")" << endl;
	cout << " F(" << x_k1 << "; " << y_k1 << ") = " << FuncXY(x_k1, y_k1) <<
endl;
	cout << " Количество итераций: " << iteration << endl;
}

void NewtonMethodApprox(double x0, double y0)
{
	vector<double> root;
	int iteration = 0;
	double x_k = x0;
	double x_k1 = x_k;
```

```
	double y_k = y0;
	double y_k1 = y_k;
	double delta = 1;
	double h = 0.0001;
	//Метод основан на матрице обратной к матрице Якоби
	//Значение Якобиана
	while (delta >= eps)
	{
		x_k1 = x_k;
		y_k1 = y_k;


		double F1dx = (F1(x_k + h, y_k) - F1(x_k, y_k)) / h;
		double F1dy = (F1(x_k, y_k + h) - F1(x_k, y_k)) / h;
		double F2dx = (F2(x_k + h, y_k) - F2(x_k, y_k)) / h;
		double F2dy = (F2(x_k, y_k + h) - F2(x_k, y_k)) / h;
		double det = F1dx * F2dy - F1dy * F2dx;

		x_k = x_k1 - (1.0 / det) * (F2dy * F1(x_k1, y_k1) - F1dy * F2(x_k1,
y_k1));
		y_k = y_k1 - (1.0 / det) * (F1dx * F2(x_k1, y_k1) - F2dx * F1(x_k1,
y_k1));

		iteration++;
		delta = fmax(abs(x_k1 - x_k), abs(y_k1 - y_k));
		cout << iteration << ": F(" << x_k << "; " << y_k << ") = " <<
FuncXY(x_k, y_k) << endl;

	}
	cout << "Точка минимума: (" << x_k1 << "; " << y_k1 << ")" << endl;
	cout << " F(" << x_k1 << "; " << y_k1 << ") = " << FuncXY(x_k1, y_k1) <<
endl;
	cout << " Количество итераций: " << iteration << endl;
}
int main()
{
	setlocale(LC_ALL, "Russian");
	vector<double> root;
	double x0 = -1;
	double y0 = 1;

	SimpleIterationsMethod(x0, y0);
	cout << "\n_____________________________\n";
	NewtonMethodExact(x0, y0);
	cout << "\n_____________________________\n";
	NewtonMethodApprox(x0, y0);
	return 0;
}
```