Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Лабораторные работы по курсу «Численные методы»

Студент:	И. С. Своеволин
Преподаватель:	Д. Е. Пивоваров
Группа:	М8О-408Б-20
Дата:	
Оценка:	
Подпись:	

Москва, 2024

1 Методы решения нелинейных уравнений и систем нелинейных уравнений

1 Постановка задачи

2.1. Реализовать методы простой итерации и Ньютона решения нелинейных уравнений в виде программ, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения найти положительный корень нелинейного уравнения (начальное приближение определить графически). Проанализировать зависимость погрешности вычислений от количества итераций.

Вариант: 20

$$tgx - 5x^2 + 1 = 0 \tag{1}$$

2 Результаты работы

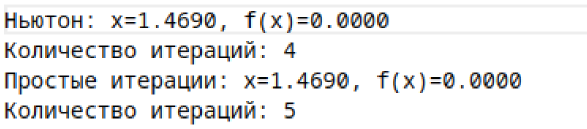

Рис. 1: Вывод в консоли

3 Исходный код

matrix.h

```
#pragma once
#include <iostream>
#include <vector>
#include <ccomplex>
#include <fstream>

using namespace std;

using cmd = complex <double>;
const double pi = acos(-1);

struct matrix
{
    int rows = 0, cols = 0;
    vector <vector <double>> v;

    matrix() {}
    matrix(int _rows, int _cols)
    {
        rows = _rows;
        cols = _cols;
        v = vector <vector <double>>(rows, vector <double>(cols));
    }

    vector <double>& operator[](int row)
    {
        return v[row];
    }

    operator double()
    {
        return v[0][0];
```

```
    }
};

matrix operator*(matrix lhs, matrix rhs)
{
    if (lhs.cols != rhs.rows)
        return matrix(0, 0);
    matrix res(lhs.rows, rhs.cols);
    for (int i = 0; i < res.rows; i++)
    {
        for (int j = 0; j < res.cols; j++)
        {
            res[i][j] = 0;
            for (int k = 0; k < lhs.cols; k++)
                res[i][j] += lhs[i][k] * rhs[k][j];
        }
    }
    return res;
}

matrix operator*(double lhs, matrix rhs)
{
    for (int i = 0; i < rhs.rows; i++)
    {
        for (int j = 0; j < rhs.cols; j++)
            rhs[i][j] *= lhs;
    }
    return rhs;
}

matrix operator+(matrix lhs, matrix rhs)
{
    if (lhs.rows != rhs.rows || rhs.cols != lhs.cols)
        return matrix(0, 0);
    matrix res(lhs.rows, lhs.cols);
    for (int i = 0; i < rhs.rows; i++)
    {
        for (int j = 0; j < res.cols; j++)
            res[i][j] = lhs[i][j] + rhs[i][j];
    }
    return res;
}

matrix operator-(matrix lhs, matrix rhs)
{
    if (lhs.rows != rhs.rows || rhs.cols != lhs.cols)
        return matrix(0, 0);
    matrix res(lhs.rows, lhs.cols);
    for (int i = 0; i < rhs.rows; i++)
```

```
    {
        for (int j = 0; j < res.cols; j++)
            res[i][j] = lhs[i][j] - rhs[i][j];
    }
    return res;
}

ostream& operator<<(ostream& stream, matrix a)
{
    for (int i = 0; i < a.rows; i++)
    {
        for (int j = 0; j < a.cols; j++)
            stream << a[i][j] << ' ';
        stream << '\n';
    }
    return stream;
}

istream& operator>>(istream& stream, matrix& a)
{
    for (int i = 0; i < a.rows; i++)
    {
        for (int j = 0; j < a.cols; j++)
            stream >> a[i][j];
    }
    return stream;
}

matrix transposition(matrix a)
{
    matrix res(a.cols, a.rows);
    for (int i = 0; i < a.rows; i++)
    {
        for (int j = 0; j < a.cols; j++)
            res[j][i] = a[i][j];
    }
    return res;
}

vector <int> swp;

pair <matrix, matrix> lu_decomposition(matrix a)
{
    int n = a.rows;
    matrix l(n, n);
    swp = vector <int>(0);
    for (int k = 0; k < n; k++)
    {
        matrix prev = a;
```

```
        int idx = k;
        for (int i = k + 1; i < n; i++)
        {
            if (abs(prev[idx][k]) < abs(prev[i][k]))
                idx = i;
        }
        swap(prev[k], prev[idx]);
        swap(a[k], a[idx]);
        swap(l[k], l[idx]);
        swp.push_back(idx);
        for (int i = k + 1; i < n; i++)
        {
            double h = prev[i][k] / prev[k][k];
            l[i][k] = h;
            for (int j = k; j < n; j++)
                a[i][j] = prev[i][j] - h * prev[k][j];

        }
    }
    for (int i = 0; i < n; i++)
        l[i][i] = 1;
    return { l, a };
}

matrix solve_triag(matrix a, matrix b, bool up)
{
    int n = a.rows;
    matrix res(n, 1);
    int d = up ? -1 : 1;
    int first = up ? n - 1 : 0;
    for (int i = first; i < n && i >= 0; i += d)
    {
        res[i][0] = b[i][0];
        for (int j = 0; j < n; j++)
        {
            if (i != j)
                res[i][0] -= a[i][j] * res[j][0];
        }
        res[i][0] = res[i][0] / a[i][i];
    }
    return res;
}

matrix solve_gauss(pair <matrix, matrix> lu, matrix b)
{
    for (int i = 0; i < swp.size(); i++)
        swap(b[i], b[swp[i]]);
    matrix z = solve_triag(lu.first, b, false);
    matrix x = solve_triag(lu.second, z, true);
```

```
    //for (int i = 0; i < swp.size(); i++)
        //swap(x[i], x[swp[i]]);
    return x;
}

matrix inverse(matrix a)
{
    int n = a.rows;
    matrix b(n, 1);
    pair <matrix, matrix> lu = lu_decomposition(a);
    matrix res(n, n);
    for (int i = 0; i < n; i++)
    {
        b[max(i - 1, 0)][0] = 0;
        b[i][0] = 1;
        matrix col = solve_gauss(lu, b);
        for (int j = 0; j < n; j++)
            res[j][i] = col[j][0];
    }
    return res;
}

double determinant(matrix a)
{
    int n = a.rows;
    pair <matrix, matrix> lu = lu_decomposition(a);
    double det = 1;
    for (int i = 0; i < n; i++)
        det *= lu.second[i][i];
    return det;
}

matrix solve_tridiagonal(matrix& a, matrix& b)
{
    int n = a.rows;
    vector <double> p(n), q(n);
    p[0] = -a[0][1] / a[0][0];
    q[0] = b[0][0] / a[0][0];
    for (int i = 1; i < n; i++)
    {
        if (i != n - 1)
            p[i] = -a[i][i + 1] / (a[i][i] + a[i][i - 1] * p[i - 1]);
        else
            p[i] = 0;
        q[i] = (b[i][0] - a[i][i - 1] * q[i - 1]) / (a[i][i] + a[i][i - 1] * p[i - 1]);
    }
    matrix res(n, 1);
    res[n - 1][0] = q[n - 1];
    for (int i = n - 2; i >= 0; i--)
```

```
        res[i][0] = p[i] * res[i + 1][0] + q[i];
    return res;
}

double abs(matrix a)
{
    double mx = 0;
    for (int i = 0; i < a.rows; i++)
    {
        double s = 0;
        for (int j = 0; j < a.cols; j++)
            s += abs(a[i][j]);
        mx = max(mx, s);
    }
    return mx;
}

matrix solve_iteration(matrix a, matrix b, double eps)
{
    int n = a.rows;
    matrix alpha(n, n), beta(n, 1);
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
            alpha[i][j] = -a[i][j] / a[i][i];
        alpha[i][i] = 0;
    }
    for (int i = 0; i < n; i++)
        beta[i][0] = b[i][0] / a[i][i];
    matrix x = beta;
    double m = abs(a);
    double epsk = 2 * eps;
    while (epsk > eps)
    {
        matrix prev = x;
        x = beta + alpha * x;
        if (m < 1)
            epsk = m / (1 - m) * abs(x - prev);
        else
            epsk = abs(x - prev);
    }
    return x;
}

matrix solve_seidel(matrix a, matrix b, double eps)
{
    int n = a.rows;
    matrix alpha(n, n), beta(n, 1);
    for (int i = 0; i < n; i++)
```

```
    {
        for (int j = 0; j < n; j++)
            alpha[i][j] = -a[i][j] / a[i][i];
        alpha[i][i] = 0;
    }
    for (int i = 0; i < n; i++)
        beta[i][0] = b[i][0] / a[i][i];
    matrix x = beta;
    double m = abs(alpha);
    double epsk = 2 * eps;
    while (epsk > eps)
    {
        matrix prev = x;
        for (int i = 0; i < n; i++)
        {
            double cur = beta[i][0];
            for (int j = 0; j < n; j++)
                cur += alpha[i][j] * x[j][0];
            x[i][0] = cur;
        }
        if (m < 1)
            epsk = m / (1 - m) * abs(x - prev);
        else
            epsk = abs(x - prev);
    }
    return x;
}

pair <matrix, matrix> method_jacobi(matrix a, double eps)
{
    int n = a.rows;
    double epsk = 2 * eps;
    matrix vec(n, n);
    for (int i = 0; i < n; i++)
        vec[i][i] = 1;
    while (epsk > eps)
    {
        int cur_i = 1, cur_j = 0;
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < i; j++)
            {
                if (abs(a[cur_i][cur_j]) < abs(a[i][j]))
                {
                    cur_i = i;
                    cur_j = j;
                }
            }
        }
```

```
        matrix u(n, n);
        double phi = pi / 4;
        if (abs(a[cur_i][cur_i] - a[cur_j][cur_j]) > 1e-7)
            phi = 0.5 * atan((2 * a[cur_i][cur_j]) / (a[cur_i][cur_i] - a[cur_j][cur_j
                ]));
        for (int i = 0; i < n; i++)
            u[i][i] = 1;
        u[cur_i][cur_j] = -sin(phi);
        u[cur_i][cur_i] = cos(phi);
        u[cur_j][cur_i] = sin(phi);
        u[cur_j][cur_j] = cos(phi);
        vec = vec * u;
        a = transposition(u) * a * u;
        epsk = 0;
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < i; j++)
                epsk += a[i][j] * a[i][j];
        }
        epsk = sqrt(epsk);
    }
    matrix val(n, 1);
    for (int i = 0; i < n; i++)
        val[i][0] = a[i][i];
    return { val, vec };
}

double sign(double x)
{
    return x > 0 ? 1 : -1;
}

pair <matrix, matrix> qr_decomposition(matrix a)
{
    int n = a.rows;
    matrix e(n, n);
    for (int i = 0; i < n; i++)
        e[i][i] = 1;
    matrix q = e;
    for (int i = 0; i < n - 1; i++)
    {
        matrix v(n, 1);
        double s = 0;
        for (int j = i; j < n; j++)
            s += a[j][i] * a[j][i];
        v[i][0] = a[i][i] + sign(a[i][i]) * sqrt(s);
        for (int j = i + 1; j < n; j++)
            v[j][0] = a[j][i];
        matrix h = e - (2.0 / double(transposition(v) * v)) * (v * transposition(v));
```

```
        q = q * h;
        a = h * a;
    }
    return { q, a };
}

vector <cmd> qr_eigenvalues(matrix a, double eps)
{
    int n = a.rows;
    vector <cmd> prev(n);
    while (true)
    {
        pair <matrix, matrix> p = qr_decomposition(a);
        a = p.second * p.first;
        vector <cmd> cur;
        for (int i = 0; i < n; i++)
        {
            if (i < n - 1 && abs(a[i + 1][i]) > 1e-7)
            {
                double b = -(a[i][i] + a[i + 1][i + 1]);
                double c = a[i][i] * a[i + 1][i + 1] - a[i][i + 1] * a[i + 1][i];
                double d = b * b - 4 * c;
                cmd sgn = (d > 0) ? cmd(1, 0) : cmd(0, 1);
                d = sqrt(abs(d));
                cur.push_back(0.5 * (-b - sgn * d));
                cur.push_back(0.5 * (-b + sgn * d));
                i++;
            }
            else
                cur.push_back(a[i][i]);
        }
        bool ok = true;
        for (int i = 0; i < n; i++)
            ok = ok && abs(cur[i] - prev[i]) < eps;
        if (ok)
            break;
        prev = cur;
    }
    return prev;
}
```

2-1.cpp

```
#include <iostream>
#include <vector>
#include <cmath>
#include <fstream>
#include <functional>
#include "matrix.h"
```

```

using namespace std;

using double_n = vector <double (*)(vector <double>)>;
using double_nn = vector <vector <double (*)(vector <double>)>>;
int iter = 0;

double dichotomy(function <double(double)> f, double l, double r, double eps)
{
    while (abs(l - r) > eps)
    {
        double nl = l + 0.3 * (r - l);
        double nr = r - 0.3 * (r - l);
        if (f(nl) < f(nr))
            l = nl;
        else
            r = nr;
    }
    return (l + r) / 2;
}

double newton(double (*f)(double), double (*df)(double), double x, double eps)
{
    double xk = x - f(x) / df(x);
    iter = 0;
    while (abs(xk - x) > eps)
    {
        x = xk;
        xk = x - f(x) / df(x);
        iter++;
    }
    return xk;
}

double iteration(double (*phi)(double), double (*dphi)(double), double l, double r,
    double eps)
{
    double x = (l + r) / 2;
    double xk = phi(x);
    iter = 0;
    double q = dphi(dichotomy(dphi, l, r, eps));
    while ((q * abs(xk - x) / (1 - q)) > eps)
    {
        x = xk;
        xk = phi(x);
        iter++;
    }
    return xk;
}
```

```
double f(double x)
{
    return tan(x) - 5. * pow(x, 2) + 1;
}

double df(double x)
{
    return (1. / pow(cos(x), 2)) - 10. * x;
}

double phi(double x)
{
    return atan(5. * pow(x, 2) - 1.);
}

double dphi(double x)
{
    return (10*x) / (pow((5 * pow(x, 2) - 1), 2) + 1);
}

int main()
{
    setlocale(LC_ALL, "Rus");
    ofstream fout("answer2-1.txt");
    fout.precision(4);
    fout << fixed;
    double ans = newton(f, df, 1.5, 0.00001);
    fout << "Ньютон: x=" << ans << ", f(x)=" << f(ans) << '\n';
    fout << "Количество итераций: " << iter << '\n';

    ans = iteration(phi, dphi, 1, 2, 0.00001);
    fout << "Простые итерации: x=" << ans << ", f(x)=" << f(ans) << '\n';
    fout << "Количество итераций: " << iter << '\n';
}
```

4 Постановка задачи

2.2. Реализовать методы простой итерации и Ньютона решения систем нелинейных уравнений в виде программного кода, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения решить систему нелинейных уравнений (при наличии нескольких решений найти то из них, в котором значения неизвестных являются положительными); начальное приближение определить графически. Проанализировать зависимость погрешности вычислений от количества итераций.

Вариант: 20

$$\begin{cases} x_1^2 - 2lgx_2 - 1 = 0 \\ x_1^2 - 2x_1 * x_2 + 2 = 0 \end{cases} \tag{2}$$

5 Результаты работы

```
Система методом Ньютона: x1=1.1488, x2=1.4449, f1(x)=0.0000, f2(x)=0.0000
Количество итераций: 4
Система методом простых итераций: x1=1.1488, x2=1.4449, f1(x)=0.0000, f2(x)=-0.0000
Количество итераций: 8
```

Рис. 2: Вывод в консоли

6 Исходный код

2-2.cpp

```
#include <iostream>
#include <vector>
#include <cmath>
#include <fstream>
#include <functional>
#include "matrix.h"

using namespace std;

using double_n = vector <double (*)(vector <double>)>;
using double_nn = vector <vector <double (*)(vector <double>)>>;
int iter = 0;

double dichotomy(function <double(double)> f, double l, double r, double eps)
{
    while (abs(l - r) > eps)
    {
        double nl = l + 0.3 * (r - l);
        double nr = r - 0.3 * (r - l);
        if (f(nl) < f(nr))
            l = nl;
        else
            r = nr;
    }
    return (l + r) / 2;
}

vector <double> newton_system(double_n f, double_nn df, vector <double> x, double eps)
```

```
{
    int n = f.size();
    matrix x0(n, 1);
    for (int i = 0; i < n; i++)
        x0[i][0] = x[i];
    matrix xk = x0;
    iter = 0;
    do
    {
        iter++;
        x0 = xk;
        for (int i = 0; i < n; i++)
            x[i] = x0[i][0];
        matrix a(n, n);
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
                a[i][j] = df[i][j](x);
        }
        matrix b(n, 1);
        for (int i = 0; i < n; i++)
            b[i][0] = -f[i](x);
        xk = x0 + solve_gauss(lu_decomposition(a), b);
    } while (abs(xk - x0) > eps);
    vector <double> res(n);
    for (int i = 0; i < n; i++)
        res[i] = xk[i][0];
    return res;
}

vector <double> iteration_system(double_n phi, double_nn dphi, vector <double> l,
    vector <double> r, double eps)
{
    double n = phi.size();
    vector <double> xk(n), x(n);
    for (int i = 0; i < n; i++)
        xk[i] = (l[i] + r[i]) / 2;
    vector <double> qn = xk;
    auto dphinx = [&](vector <double> x)
    {
        double res = 0;
        for (int j = 0; j < n; j++)
        {
            double tmp = 0;
            for (int k = 0; k < n; k++)
                tmp += abs(dphi[j][k](x));
            res = max(res, tmp);
        }
        return res;
```

```
    };
    for (int i = 0; i < n; i++)
    {
        auto dphix = [&](double x)
        {
            vector <double> xn = qn;
            xn[i] = x;
            return dphinx(xn);
        };
        qn[i] = dichotomy(dphix, l[i], r[i], eps);
    }
    double q = dphinx(qn);
    double epsk = 0;
    iter = 0;
    do
    {
        iter++;
        x = xk;
        for (int i = 0; i < n; i++)
            xk[i] = phi[i](x);
        epsk = 0;
        for (int i = 0; i < n; i++)
        {
            double s = abs(x[i] - xk[i]);
            epsk = max(epsk, s);
        }
    } while (q * epsk / (1 - q) > eps);
    return xk;
}

double f1(vector <double> x)
{
    return x[0] * x[0] - 2. * log10(x[1]) - 1.;
}

double f2(vector <double> x)
{
    return x[0] * x[0] - 2. * x[0] * x[1] + 2.;
}

double_n fn = { f1, f2 };

double df11(vector <double> x)
{
    return 2. * x[0];
}
double df12(vector <double> x)
{
    return - 2. / (x[1] * log(10));
```

```
}

double df21(vector <double> x)
{
    return 2 * x[0] - 2 * x[1];
}

double df22(vector <double> x)
{
    return - 2 * x[0];
}

double_nn dfn = { {df11, df12}, {df21, df22} };

double phi1(vector <double> x)
{
    return sqrt(2. * log10(x[1]) + 1);
}

double phi2(vector <double> x)
{
    return (pow(x[0], 2) + 2) / (2 * x[0]);
}

double_n phin = { phi1, phi2 };

double dphi11(vector <double> x)
{
    return 0;
}

double dphi12(vector <double> x)
{
    return pow((x[1] * sqrt(log(10))) * sqrt(2*log(x[1]) + log(10)), -1);
}

double dphi21(vector <double> x)
{
    return pow(2, -1) - pow(x[0], -2);
}

double dphi22(vector <double> x)
{
    return 0;
}

double_nn dphin = { {dphi11, dphi12}, {dphi21, dphi22} };

int main()
```

```
{
    setlocale(LC_ALL, "Rus");
    ofstream fout("answer2-2.txt");
    fout.precision(4);
    fout << fixed;
    vector <double> x = newton_system(fn, dfn, { 1, 1 }, 0.00001);
    fout << "Система методомНьютона: x1=" << x[0] << ", x2=" << x[1] << ", f1(x)=" <<
        f1(x) << ", f2(x)=" << f2(x) << '\n';
    fout << "Количество итераций: " << iter << '\n';

    x = iteration_system(phin, dphin, { 1, 1 }, { 2, 2 }, 0.00001);
    fout << "Система методомпростыхитераций: x1=" << x[0] << ", x2=" << x[1] << ", f1(x
        )=" << f1(x) << ", f2(x)=" << f2(x) << '\n';
    fout << "Количество итераций: " << iter << '\n';
}
```