```
/*
 * Ising_Magnetisierung.cpp
 *
 *  Created on: 12.06.2017
 *      Author: mona
 */

#include <iostream>
#include <string>
#include <cstdlib>
#include <complex>
#include <vector>
#include <cmath>
#include <sstream>
#include <utility>
#include <math.h>
#include <fstream>
#include <functional>
#include "Eigen/Dense"
#include "Eigen/Core"

using namespace std;
using namespace Eigen;


double Funktion(double h, double t, double m){
        double F=m-tanh((h+m)/t);
        return F;
}

double Startwert_select(double h, double t){
        //cout << "startwert wird gesucht" << endl;
        double prod=1;
        double x0;
        double y0;
        while (prod>=0){
                x0=rand()* (1./RAND_MAX);
                y0=-rand()* (1./RAND_MAX);
                prod=Funktion(h,t,x0)*Funktion(h,t,y0);
        }
        double x=x0;
        double y=y0;
        double krit=0.0001;

        while(Funktion(h,t,x)>krit){
                double z=(x+y)/2;
                if(Funktion(h,t,z)*Funktion(h,t,x)<0){y=z;}
                else{x=z;}
        }
        return x;
}

double Newton_Raphson_Step(double m, double h, double t){

        double f=m-tanh((h+m)/t);
        double sech_quad=cosh(2*(h+m)/t)+1;
        sech_quad=2/sech_quad;
        double f_strich=1-(1/t)*sech_quad;

        double m_new=m-(f/f_strich);

        return m_new;
}

double Newton_Raphson_Iterate(double m0, double h, double t){

        double m_krit=0.0001;
        double m_new=0;
        double diff=1;
        double m=m0;
        int counter=0;
        int counter_max=50;

        while(diff>m_krit && counter < counter_max){
                m_new=Newton_Raphson_Step(m, h, t);
                diff=abs(m_new-m);
                m=m_new;
                counter ++;
                if (counter==counter_max){
                        cout << "Achtung, wahrscheinlich keine Konvergenz der Iteration!" << endl;}
        }
        //cout << "counter" << counter << endl;
        return m;
}

```

```
void Newton_Raphson_tvari( double h, string Name ){

        double t_start=0.01;
        double t_max=3;
        double tstep=0.001;

        // ==============================
        ofstream Data;
        Data.open(Name.c_str());
        Data.precision(10);//
        // ==============================

        for(double t=t_start; t<t_max; t=t+tstep){
                for(int i=1; i<5; i++){
                //Gucke fuer 4 unterschiedliche Startwerte nach Fixpunkten
                double m0=Startwert_select(h, t);
                double m=Newton_Raphson_Iterate(m0, h, t);
                Data << t << "\t" << m << endl;
                }
        }

}

void Newton_Raphson_hvari2(double t, string Name ){

        double h_start=-3.01;
        double h_max=3;
        double hstep=0.01;
        double hstep2=0.00001;

        // ==============================
        ofstream Data;
        Data.open(Name.c_str());
        Data.precision(10);//
        // ==============================

        for(double h=h_start; h<-0.9; h=h+hstep){
                //double m=Newton_Raphson_Iterate(0, h, t);
                //Data << h << "\t" << m << endl;
                for(int i=1; i<4; i++){
                double m0=Startwert_select(h, t);
                double m1=Newton_Raphson_Iterate(m0, h, t);
                Data << h << "\t" << m1 << endl;
                //cout << h << "\t" << m1 << endl;
                }
        }

        //Im Bereich von -1 bis 1 kommen besonders viele Fixpunkte in Abhaengigkeit des Startwertes vor
        //daher verkleinern wir hier die Schritteweite
        for(double h=-0.9; h<0.9; h=h+hstep2){
                //double m=Newton_Raphson_Iterate(0, h, t);
                //Data << h << "\t" << m << endl;
                for(int i=1; i<4; i++){
                double m0=Startwert_select(h, t);
                double m1=Newton_Raphson_Iterate(m0, h, t);
                Data << h << "\t" << m1 << endl;
                //cout << h << "\t" << m1 << endl;
                }
        }

        for(double h=0.9; h<h_max; h=h+hstep){
                //double m=Newton_Raphson_Iterate(0, h, t);
                //Data << h << "\t" << m << endl;
                for(int i=1; i<4; i++){
                double m0=Startwert_select(h, t);
                double m1=Newton_Raphson_Iterate(m0, h, t);
                Data << h << "\t" << m1 << endl;
                //cout << h << "\t" << m1 << endl;
                }
        }

}


void Newton_Raphson_hvari(double t, string Name ){

        double h_start=-3.01;
        double h_max=3;
        double hstep=0.001;

        // ==============================
        ofstream Data;
        Data.open(Name.c_str());
        Data.precision(10);//
        // ==============================

```

```
        for(double h=h_start; h<h_max; h=h+hstep){
                //double m=Newton_Raphson_Iterate(0, h, t);
                //Data << h << "\t" << m << endl;
                for(int i=1; i<4; i++){
                double m0=Startwert_select(h, t);
                double m1=Newton_Raphson_Iterate(m0, h, t);
                Data << h << "\t" << m1 << endl;
                //cout << h << "\t" << m1 << endl;
                }
        }


}




int main(){

        double h;
        double t;

        cout << "h=0: " << endl;
        h=0.0;
        Newton_Raphson_tvari(h, "Data_h0" );
        cout << "h=01: " << endl;
        h=0.1;
        Newton_Raphson_tvari( h, "Data_h01" );
        cout << "h=05: " << endl;
        h=0.5;
        Newton_Raphson_tvari(h, "Data_h05" );

        cout << "t=10: " << endl;
        t=1.0;
        Newton_Raphson_hvari(t, "Data_t10" );
        cout << "t=15: " << endl;
        t=1.5;
        Newton_Raphson_hvari(t, "Data_t15" );
        cout << "t=05: " << endl;

        t=0.5;
        Newton_Raphson_hvari2( t, "Data_t05" );

        cout << "Ende der main-Funktion" << endl;
        return 0;
}
```

```
/*
 * Bifurkation.cpp
 *
 *  Created on: 12.06.2017
 *      Author: mona
 */

#include <iostream>
#include <string>
#include <cstdlib>
#include <complex>
#include <vector>
#include <cmath>
#include <sstream>
#include <utility>
#include <math.h>
#include <fstream>
#include <functional>
#include "Eigen/Dense"
#include "Eigen/Core"

using namespace std;
using namespace Eigen;


double Step_log_Abb(double x1, double r){
        double x2;

        if(x1<0 or x1>1){
                cout << "x liegt ausserhalb des Intervalls [0,1]!" << endl;
                return 0.0;
        }

        x2=r*x1*(1-x1);
        return x2;
}

double Step_kub_Abb(double x1, double r){
        double x2;

        if(x1<(-sqrt(1+r)) or x1>(sqrt(1+r))){
                cout << "x liegt ausserhalb des Intervalls [-sqrt(1+r),sqrt(1+r)]!" << endl;
                cout << "x1= " << x1 << "    r= " << r << endl;
                return 0.0;
        }


        x2=r*x1-x1*x1*x1;
        return x2;
}

void iteration(double (*F)(double, double), double rmax , double delta_r, string Name, bool kubisch){
        int kalibrierung=300;
        int auswertungen=60;

        // ===============================
        ofstream Data;
        Data.open(Name.c_str());
        Data.precision(10);//
        // ===============================

        for (double r=0; r<rmax; r=r+delta_r){
                for (int j=0; j<auswertungen; j++){
                        double x0=0.0;
                        if(kubisch==true){
                                x0=rand()* (1./RAND_MAX)*sqrt(1+r)*2 ;
                                x0=x0-sqrt(1+r);
                        }
                        else{
                                x0=rand()* (1./RAND_MAX);
                        }

                        for(int i=0; i<kalibrierung; i++){
                        x0=F(x0, r);}

                        Data << r << "\t" << x0 << endl;

                }
        }


}


int main(){
        //Fuer die logistische Abbildung divergiert der wert der iteration fuer r>4
```

```
        iteration(Step_log_Abb, 4.0 , 0.001, "logistische_Abb", false);
        cout << "Ab hier kubische Abbildung" << endl;
        //Fuer die kubische Abbildung divergiert der wert der iteration fuer r>3
        iteration(Step_kub_Abb, 3.00 , 0.001, "kubische_Abb", true);

        cout << "Ende der main-Funktion" << endl;
        return 0;
}
```