```cpp
/*
 * 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*2;
                y0=-rand()*1./RAND_MAX*2;
                //cout << "x0= " << x0 << '\t' << "y0= " << y0 << endl;
                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 liegen mehrere Fixpunkte vor!
" << endl;}
        }
        //cout << "counter" << counter << endl;
        return m;
}
```

```
 86:
 87: void Newton_Raphson_tvari( double h, string Name ){
 88:
 89:         double t_start=0.01;
 90:         double t_max=3;
 91:         double tstep=0.001;
 92:
 93:         // ================================
 94:         ofstream Data;
 95:         Data.open(Name.c_str());
 96:         Data.precision(10);//
 97:         // ================================
 98:
 99:         for(double t=t_start; t<t_max; t=t+tstep){
100:                 double m0=Startwert_select(h, t);
101:                 double m=Newton_Raphson_Iterate(m0, h, t);
102:                 Data << t << "\t" << m << endl;
103:         }
104:
105: }
106:
107: void Newton_Raphson_hvari(double t, string Name ){
108:
109:         double h_start=-3.01;
110:         double h_max=3;
111:         double hstep=0.00001;
112:
113:         // ================================
114:         ofstream Data;
115:         Data.open(Name.c_str());
116:         Data.precision(10);//
117:         // ================================
118:
119:         for(double h=h_start; h<h_max; h=h+hstep){
120:                 //cout << "beginn der for-Schleife" << endl;
121:                 double m0=Startwert_select(h, t);
122:                 double m=Newton_Raphson_Iterate(m0, h, t);
123:                 Data << h << "\t" << m << endl;
124:         }
125:
126: }
127:
128:
129:
130:
131: int main(){
132:
133:         double h;
134:         double t;
135:
136:         cout << "h=0: " << endl;
137:         h=0.0;
138:         Newton_Raphson_tvari(h, "Data_h0" );
139:         cout << "h=01: " << endl;
140:         h=0.1;
141:         Newton_Raphson_tvari( h, "Data_h01" );
142:         cout << "h=05: " << endl;
143:         h=0.5;
144:         Newton_Raphson_tvari(h, "Data_h05" );
145:
146:         cout << "t=10: " << endl;
147:         t=1.0;
148:         Newton_Raphson_hvari(t, "Data_t10" );
149:         cout << "t=15: " << endl;
150:         t=1.5;
151:         Newton_Raphson_hvari(t, "Data_t15" );
152:         cout << "t=05: " << endl;
153:         t=0.5;
154:         Newton_Raphson_hvari( t, "Data_t05" );
155:
156:         cout << "Ende der main-Funktion" << endl;
157:         return 0;
158: }
```

```cpp
/*
 * 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;
}

```

```
 87: void Newton_Raphson_tvari( double h, string Name ){
 88:
 89:         double t_start=0.01;
 90:         double t_max=3;
 91:         double tstep=0.001;
 92:
 93:         // ================================
 94:         ofstream Data;
 95:         Data.open(Name.c_str());
 96:         Data.precision(10);//
 97:         // ================================
 98:
 99:         for(double t=t_start; t<t_max; t=t+tstep){
100:                 for(int i=1; i<5; i++){
101:                 //Gucke fuer 4 unterschiedliche Startwerte nach Fixpunkten
102:                 double m0=Startwert_select(h, t);
103:                 double m=Newton_Raphson_Iterate(m0, h, t);
104:                 Data << t << "\t" << m << endl;
105:                 }
106:         }
107:
108: }
109:
110: void Newton_Raphson_hvari2(double t, string Name ){
111:
112:         double h_start=-3.01;
113:         double h_max=3;
114:         double hstep=0.01;
115:         double hstep2=0.00001;
116:
117:         // ================================
118:         ofstream Data;
119:         Data.open(Name.c_str());
120:         Data.precision(10);//
121:         // ================================
122:
123:         for(double h=h_start; h<-0.9; h=h+hstep){
124:                 //double m=Newton_Raphson_Iterate(0, h, t);
125:                 //Data << h << "\t" << m << endl;
126:                 for(int i=1; i<4; i++){
127:                 double m0=Startwert_select(h, t);
128:                 double m1=Newton_Raphson_Iterate(m0, h, t);
129:                 Data << h << "\t" << m1 << endl;
130:                 //cout << h << "\t" << m1 << endl;
131:                 }
132:         }
133:
134:         //Im Bereich von -1 bis 1 kommen besonders viele Fixpunkte in Abhaengigkeit des Startwertes vor
135:         //daher verkleinern wir hier die Schritteweite
136:         for(double h=-0.9; h<0.9; h=h+hstep2){
137:                 //double m=Newton_Raphson_Iterate(0, h, t);
138:                 //Data << h << "\t" << m << endl;
139:                 for(int i=1; i<4; i++){
140:                 double m0=Startwert_select(h, t);
141:                 double m1=Newton_Raphson_Iterate(m0, h, t);
142:                 Data << h << "\t" << m1 << endl;
143:                 //cout << h << "\t" << m1 << endl;
144:                 }
145:         }
146:
147:         for(double h=0.9; h<h_max; h=h+hstep){
148:                 //double m=Newton_Raphson_Iterate(0, h, t);
149:                 //Data << h << "\t" << m << endl;
150:                 for(int i=1; i<4; i++){
151:                 double m0=Startwert_select(h, t);
152:                 double m1=Newton_Raphson_Iterate(m0, h, t);
153:                 Data << h << "\t" << m1 << endl;
154:                 //cout << h << "\t" << m1 << endl;
155:                 }
156:         }
157:
158: }
159:
160:
161: void Newton_Raphson_hvari(double t, string Name ){
162:
163:         double h_start=-3.01;
164:         double h_max=3;
165:         double hstep=0.001;
166:
167:         // ================================
168:         ofstream Data;
169:         Data.open(Name.c_str());
170:         Data.precision(10);//
171:         // ================================
172:
```

```
173:         for(double h=h_start; h<h_max; h=h+hstep){
174:                 //double m=Newton_Raphson_Iterate(0, h, t);
175:                 //Data << h << "\t" << m << endl;
176:                 for(int i=1; i<4; i++){
177:                 double m0=Startwert_select(h, t);
178:                 double m1=Newton_Raphson_Iterate(m0, h, t);
179:                 Data << h << "\t" << m1 << endl;
180:                 //cout << h << "\t" << m1 << endl;
181:                 }
182:         }
183:
184:
185: }
186:
187:
188:
189:
190: int main(){
191:
192:         double h;
193:         double t;
194:
195:         cout << "h=0: " << endl;
196:         h=0.0;
197:         Newton_Raphson_tvari(h, "Data_h0" );
198:         cout << "h=01: " << endl;
199:         h=0.1;
200:         Newton_Raphson_tvari( h, "Data_h01" );
201:         cout << "h=05: " << endl;
202:         h=0.5;
203:         Newton_Raphson_tvari(h, "Data_h05" );
204:
205:         cout << "t=10: " << endl;
206:         t=1.0;
207:         Newton_Raphson_hvari(t, "Data_t10" );
208:         cout << "t=15: " << endl;
209:         t=1.5;
210:         Newton_Raphson_hvari(t, "Data_t15" );
211:         cout << "t=05: " << endl;
212:
213:         t=0.5;
214:         Newton_Raphson_hvari2( t, "Data_t05" );
215:
216:         cout << "Ende der main-Funktion" << endl;
217:         return 0;
218: }
219:
```

```cpp
/*
 * 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
```

```
87:         iteration(Step_log_Abb, 4.0 , 0.001, "logistische_Abb", false);
88:         cout << "Ab hier kubische Abbildung" << endl;
89:         //Fuer die kubische Abbildung divergiert der wert der iteration fuer r>3
90:         iteration(Step_kub_Abb, 3.00 , 0.001, "kubische_Abb", true);
91:
92:         cout << "Ende der main-Funktion" << endl;
93:         return 0;
94: }
```

```cpp
/*
 * 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
```

```
87:         iteration(Step_log_Abb, 4.0 , 0.001, "logistische_Abb", false);
88:         cout << "Ab hier kubische Abbildung" << endl;
89:         //Fuer die kubische Abbildung divergiert der wert der iteration fuer r>3
90:         iteration(Step_kub_Abb, 3.00 , 0.001, "kubische_Abb", true);
91:
92:         cout << "Ende der main-Funktion" << endl;
93:         return 0;
94: }
```