

```

1:  /*
2:   * Ising_Magnetisierung.cpp
3:   *
4:   * Created on: 12.06.2017
5:   * Author: mona
6:   */
7:
8: #include <iostream>
9: #include <string>
10: #include <cstdlib>
11: #include <complex>
12: #include <vector>
13: #include <cmath>
14: #include <sstream>
15: #include <utility>
16: #include <math.h>
17: #include <fstream>
18: #include <functional>
19: #include "Eigen/Dense"
20: #include "Eigen/Core"
21:
22: using namespace std;
23: using namespace Eigen;
24:
25:
26:
27: double Funktion(double h, double t, double m){
28:     double F=m-tanh((h+m)/t);
29:     return F;
30: }
31:
32: double Startwert_select(double h, double t){
33:     //cout << "startwert wird gesucht" << endl;
34:     double prod=1;
35:     double x0;
36:     double y0;
37:     while (prod>=0){
38:         x0=rand()*(1./RAND_MAX);
39:         y0=-rand()*(1./RAND_MAX);
40:         prod=Funktion(h,t,x0)*Funktion(h,t,y0);
41:     }
42:     double x=x0;
43:     double y=y0;
44:     double krit=0.0001;
45:
46:     while (Funktion(h,t,x)>krit){
47:         double z=(x+y)/2;
48:         if (Funktion(h,t,z)*Funktion(h,t,x)<0){y=z;}
49:         else{x=z;}
50:     }
51:     return x;
52: }
53:
54: double Newton_Raphson_Step(double m, double h, double t){
55:
56:     double f=m-tanh((h+m)/t);
57:     double sech_quad=cosh(2*(h+m)/t)+1;
58:     sech_quad=2/sech_quad;
59:     double f_strich=1-(1/t)*sech_quad;
60:
61:     double m_new=m-(f/f_strich);
62:
63:     return m_new;
64: }
65:
66: double Newton_Raphson_Iterate(double m0, double h, double t){
67:
68:     double m_krit=0.0001;
69:     double m_new=0;
70:     double diff=1;
71:     double m=m0;
72:     int counter=0;
73:     int counter_max=50;
74:
75:     while(diff>m_krit && counter < counter_max){
76:         m_new=Newton_Raphson_Step(m, h, t);
77:         diff=abs(m_new-m);
78:         m=m_new;
79:         counter++;
80:         if (counter==counter_max){
81:             cout << "Achtung, wahrscheinlich keine Konvergenz der Iteration!" << endl;
82:         }
83:         //cout << "counter" << counter << endl;
84:         return m;
85:     }
86:

```

```
87: void Newton_Raphson_tvarti( 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_hvarti2(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 Schrittwerte
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_hvarti(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_tvvari(h, "Data_h0" );
198:     cout << "h=01: " << endl;
199:     h=0.1;
200:     Newton_Raphson_tvvari( h, "Data_h01" );
201:     cout << "h=05: " << endl;
202:     h=0.5;
203:     Newton_Raphson_tvvari(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:
```

```
1: /*
2:  * Bifurkation.cpp
3:  *
4:  * Created on: 12.06.2017
5:  * Author: mona
6:  */
7:
8: #include <iostream>
9: #include <string>
10: #include <cstdlib>
11: #include <complex>
12: #include <vector>
13: #include <cmath>
14: #include <sstream>
15: #include <utility>
16: #include <math.h>
17: #include <fstream>
18: #include <functional>
19: #include "Eigen/Dense"
20: #include "Eigen/Core"
21:
22: using namespace std;
23: using namespace Eigen;
24:
25:
26: double Step_log_Abb(double x1, double r){
27:     double x2;
28:
29:     if(x1<0 or x1>1){
30:         cout << "x liegt ausserhalb des Intervalls [0,1]!" << endl;
31:         return 0.0;
32:     }
33:
34:     x2=r*x1*(1-x1);
35:     return x2;
36: }
37:
38: double Step_kub_Abb(double x1, double r){
39:     double x2;
40:
41:     if(x1<(-sqrt(1+r)) or x1>(sqrt(1+r))){
42:         cout << "x liegt ausserhalb des Intervalls [-sqrt(1+r),sqrt(1+r)]!" << endl;
43:         cout << "x1= " << x1 << " r= " << r << endl;
44:         return 0.0;
45:     }
46:
47:
48:     x2=r*x1-x1*x1*x1;
49:     return x2;
50: }
51:
52: void iteration(double (*F)(double, double), double rmax, double delta_r, string Name, bool kubisch){
53:     int kalibrierung=300;
54:     int auswertungen=60;
55:
56:     // =====
57:     ofstream Data;
58:     Data.open(Name.c_str());
59:     Data.precision(10);
60:     // =====
61:
62:     for (double r=0; r<rmax; r=r+delta_r){
63:         for (int j=0; j<auswertungen; j++){
64:             double x0=0.0;
65:             if(kubisch==true){
66:                 x0=rand()* (1./RAND_MAX)*sqrt(1+r)*2 ;
67:                 x0=x0-sqrt(1+r);
68:             }
69:             else{
70:                 x0=rand()* (1./RAND_MAX);
71:             }
72:
73:             for(int i=0; i<kalibrierung; i++){
74:                 x0=F(x0, r);
75:             }
76:
77:             Data << r << "\t" << x0 << endl;
78:         }
79:     }
80:
81:
82: }
83:
84:
85: int main(){
86:     //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: }
```