```cpp
/*
 * Wettervorhersage.cpp
 *
 *  Created on: 02.05.2017
 *      Author: mona
 */

#include <iostream>
#include <iomanip>
#include <complex>
#include <cstdlib>
#include <vector>
#include <cmath>
#include <sstream>
#include <utility>
#include <math.h>
#include <fstream>
#include <functional>
using namespace std;


//***************************************
// Definition des Startvektors und des DGL-Systems
//***************************************

vector<double> DGL_start={-10,15,19}; //Vektor der Startwerte

vector<double> f(vector<double> DGL, double sigma, double r, double b){

        double fx=sigma*(-DGL[0]+DGL[1]);
        double fy=-DGL[0]*DGL[2]+r*DGL[0]-DGL[1];
        double fz=DGL[0]*DGL[1]-b*DGL[2];
        DGL={fx,fy,fz};
        return DGL;
}


//***************************************
// Aufgabenteil a) Integration des gleichungssystems mit dem Euler verfahren und Ausgabe von 3 Textdateien,
// welche die werte fÃ¼r N={10,100,1000} ausgibt.
//***************************************

void euler(vector<double> DGL, double h1, double h2, double h3, double N, double sigma, double r, double b){

        vector<double> DGL_1=DGL;
        vector<double> DGL_2=DGL;
        vector<double> DGL_3=DGL;
        // ================================
        ofstream a;
        a.open("Wetter_h1.txt");
        a.precision(10);//
        // ================================

        for(double n=1; n<=N; n++){
                vector<double> func1=f(DGL_1,sigma,r,b);
                double x=DGL_1[0]+h1*func1[0];
                double y=DGL_1[1]+h1*func1[1];
                double z=DGL_1[2]+h1*func1[2];
                if(n==10 || n==100 || n==1000){
                        a << n << "\t" << h1*n << "\t" << x << "\t" << y << "\t" <<  z << endl;
                }
                DGL_1={x,y,z};
        }
        a.close();
        // ================================
                ofstream c;
                c.open("Wetter_h2.txt");
                c.precision(10);//
        // ================================

        for(double n=1; n<=N; n++){
                vector<double> func2=f(DGL_2,sigma,r,b);
                double x=DGL_2[0]+h2*func2[0];
                double y=DGL_2[1]+h2*func2[1];
                double z=DGL_2[2]+h2*func2[2];
                if(n==10 || n==100 || n==1000){
                        c << n << "\t" << h2*n << "\t" << x << "\t" << y << "\t" <<  z << endl;
                }
                DGL_2={x,y,z};
        }
        c.close();
        // ================================
                ofstream d;
                d.open("Wetter_h3.txt");
```

```
85:                 d.precision(10);//
86:         // ================================
87:
88:         for(double n=1; n<=N; n++){
89:                 vector<double> func3=f(DGL_3,sigma,r,b);
90:                 double x=DGL_3[0]+h3*func3[0];
91:                 double y=DGL_3[1]+h3*func3[1];
92:                 double z=DGL_3[2]+h3*func3[2];
93:                 if(n==10 || n==100 || n==1000){
94:                         d << n << "\t" << h3*n << "\t" << x << "\t" << y << "\t" <<  z << endl;
95:                 }
96:                 DGL_3={x,y,z};
97:         }
98:         d.close();
99: }
100:
101: //*************************************
102: // Aufgabenteil b) und c) Ausgabe von DatensÃ¤tzen fÃ¼r x und y fÃ¼r verschiedene r und Startwerte
103: //*************************************
104:
105:
106: void euler_plot(vector<double> DGL, double h, double N, double sigma, double r1, double r2, double b){
107:
108:         vector<double> DGL_1=DGL;
109:         vector<double> DGL_2=DGL;
110:
111:         // ================================
112:         ofstream a;
113:         a.open("Wetter_plotr20.txt");
114:         a.precision(10);//
115:         // ================================
116:
117:         for(double n=1; n<=N; n++){
118:                 vector<double> func1=f(DGL_1,sigma,r1,b);
119:                 double x=DGL_1[0]+h*func1[0];
120:                 double y=DGL_1[1]+h*func1[1];
121:                 double z=DGL_1[2]+h*func1[2];
122:                 a << n << "\t" << h*n << "\t" << x << "\t" << y  << endl;
123:                 DGL_1={x,y,z};
124:         }
125:         a.close();
126:
127:         // ================================
128:         ofstream c;
129:         c.open("Wetter_plotr28.txt");
130:         c.precision(10);//
131:         // ================================
132:
133:         for(double n=1; n<=N; n++){
134:                 vector<double> func2=f(DGL_2,sigma,r2,b);
135:                 double x=DGL_2[0]+h*func2[0];
136:                 double y=DGL_2[1]+h*func2[1];
137:                 double z=DGL_2[2]+h*func2[2];
138:                 c << n << "\t" << h*n << "\t" << x << "\t" << y << endl;
139:                 DGL_2={x,y,z};
140:         }
141:         c.close();
142:
143:         // Aufgabenteil c)
144:         vector<double> DGL_3={-10.01,15,19};
145:
146:         // ================================
147:         ofstream d;
148:         d.open("Wetter_plotr20_c.txt");
149:         d.precision(10);//
150:         // ================================
151:
152:         for(double n=1; n<=N; n++){
153:                 vector<double> func3=f(DGL_3,sigma,r1,b);
154:                 double x=DGL_3[0]+h*func3[0];
155:                 double y=DGL_3[1]+h*func3[1];
156:                 double z=DGL_3[2]+h*func3[2];
157:                 d << n << "\t" << h*n << "\t" << x << "\t" << y << endl;
158:                 DGL_3={x,y,z};
159:         }
160:         d.close();
161:
162:
163: }
164:
165: //*************************************
166: // Aufgabenteil d) Numerische Ermittlung des Fixpunktes als letzten Wert
167: //*************************************
168:
169: vector<double> euler_Fixpunkt(vector<double> DGL, double h, double N, double sigma, double r, double b){
170:
```

```
  171:         vector<double> DGL_1=DGL;
  172:
  173:         double x_alt=DGL_1[0];
  174:         double y_alt=DGL_1[1];
  175:         double z_alt=DGL_1[2];
  176:         double x;
  177:         double y;
  178:         double z;
  179:
  180:         double variation=0.00001;
  181:
  182:         for(double n=1; n<=N; n++){
  183:                 vector<double> func1=f(DGL_1,sigma,r,b);
  184:                 x=DGL_1[0]+h*func1[0];
  185:                 y=DGL_1[1]+h*func1[1];
  186:                 z=DGL_1[2]+h*func1[2];
  187:                 if(abs(x_alt-x)<variation && abs(y_alt-y)<variation && abs(z_alt-z)<variation){
  188:                         cout << "fixpunkt bei x = " << x << " , y = " << y << " , z = " << z << endl;
  189:                         break;
  190:                 }
  191:                 x_alt=x;
  192:                 y_alt=y;
  193:                 z_alt=z;
  194:                 DGL_1={x,y,z};
  195:         }
  196:         vector<double> letzterWert= {x,y,z};
  197:         return letzterWert;
  198: }
  199:
  200:
  201:
  202: int main(){
  203:         double r=20;
  204:         double sigma=10;
  205:         double b= 8./3.;   //Wird sonst behandelt wie ein Integer
  206:         euler(DGL_start,0.05,0.005,0.0005,1000,sigma,r,b);
  207:         euler_plot(DGL_start,0.01,1e5,sigma,20,28,b);
  208:
  209:         //Berechnung des analytischen Fixpunktes
  210:         double Fix_20= sqrt(b*(20-1));
  211:         double Fix_28= sqrt(b*(28-1));
  212:         cout << scientific << setprecision(8)<< "Analytisch berechneter Fixpunkt bei r=20: " << "\t" <<
Fix_20  << endl;
  213:         cout << scientific << setprecision(8)<< "Analytisch berechneter Fixpunkt bei r=28: " << "\t" <<
Fix_28  << endl << endl << endl;
  214:
  215:         // Numerische berechnung der Fixpunkte
  216:         cout << scientific << setprecision(8)<< "Fixpunkt bei r=20, Start=(-10,15,19): " << "\t" ;
  217:         vector<double> letzter=euler_Fixpunkt(DGL_start,0.001,1e5,sigma,20,b);
  218:
  219:         cout << "Letzter erreichter Wert: " << "\t" << letzter[0] <<"\t" << letzter[1]<<"\t" << letzter[
2] << endl ;
  220:
  221:
  222:         //Fehler fÃ¼r x
  223:         double x_Fehler=abs(-letzter[0]-Fix_20)/Fix_20;
  224:         double y_Fehler=abs(-letzter[1]-Fix_20)/Fix_20;
  225:         cout << "Fehler fuer x: " << "\t" << x_Fehler <<"\t" << "Fehler fuer y:"<<"\t" << y_Fehler << en
dl << endl;
  226:
  227:
  228:
  229:         cout << "Fixpunkt bei r=20, Start=(10.01,15,19): " << "\t" ;
  230:         vector<double> DGL_start_neu={10.01,15,19}; //Neuer Vektor der Startwerte
  231:         letzter=euler_Fixpunkt(DGL_start_neu,0.001,1e5,sigma,20,b);
  232:         x_Fehler=abs(letzter[0]-Fix_20)/Fix_20;
  233:         y_Fehler=abs(letzter[1]-Fix_20)/Fix_20;
  234:         cout << "Letzter erreichter Wert: " << "\t" << letzter[0] <<"\t" << letzter[1]<<"\t" << letzter[
2] << endl << endl;
  235:         cout << "Fehler fuer x: " << "\t" << x_Fehler <<"\t" << "Fehler fuer y:"<<"\t" << y_Fehler << en
dl << endl;
  236:
  237:         return 0;
  238: }
```

```cpp
/*
 * A2.cpp
 *
 *  Created on: 28.04.2017
 *      Author: mona
 */


#include <iostream>
#include <iomanip>
#include <complex>
#include <cstdlib>
#include <vector>
#include <cmath>
#include <sstream>
#include <utility>
#include <math.h>
#include <fstream>
#include <functional>
using namespace std;

//***************************************
// Funktionsvorschriften
//***************************************

double F1(double x){
        return exp(x)/x;
}

double F1_sing(double x, double lim){
                        return lim+0.5*lim*lim*x;
}

double F2(double x){
        return 2 * exp(-x * x);
}

double F3(double x){
        if(x==0){
                return 1;
        }
        return sin(x)/x;
}

//***************************************
// Trapezregel
//***************************************

double trapez(double a, double b, double N, double(*f)(double)){
        double h=(b-a)/N;
        double summe=(h/2)*(f(a)+f(b));
        for (int n=1; n<N; n++){
                summe=summe+h*f(a+n*h);
        };
        return summe;
}

//*****************************************************************
// Trapezregel fuer den singulaeren Bereich um 0 fuer Funktion 2
// benoeigt ein zusaetzliches Argument in f: lim.
//*****************************************************************

double trapezlim(double a, double b, double N, double(*f)(double,double), double lim){
        double h=(b-a)/N;
        double summe=(h/2)*(f(a,lim)+f(b,lim));
        for (int n=1; n<N; n++){
                summe=summe+h*f(a+n*h,lim);
        };
        return summe;
}

//*****************************************************************
// GENAUIGKEIT
//*****************************************************************

double eps = 1e-5;

//*****************************************************************
// Funktion fuer Aufgabenteil a)
//*****************************************************************

double Trapez(double a, double b){
        double lim = 0.00001;
                double Delta = 1;
                double N=2;
                double integral1;
```

```
    87:                 int i = 0;
    88:
    89:                     while(abs(Delta)>eps){
    90:                                 ++i;
    91:                                 integral1 = trapez(a,-lim,N,F1)+trapez(lim,b,N,F1)+trapezlim(-1,1,N,F1_s
ing,lim); // berechne Integral mit halber Anzahl Trapeze
    92:                                 Delta = abs(integral1-2.1145018)/2.1145018; // relativer Fehler zum wahr
en Wert
    93:                                 N = 2*N;
    94:                                 lim=lim/2;
    95:                     }
    96:                     cout << "Anzahl der Iterationen: " << i << endl;
    97:                     cout << "Wert des Integrals 2a): " << integral1 << endl;
    98:                     return integral1;
    99: }
   100:
   101: /* Alternative: Genauigkeit nicht zum exakten wert, sondern zum vorherigen
   102:  double Trapez(double a, double b){
   103:         double lim = 0.00001;
   104:                 double Delta = 1;
   105:                 double N=2;
   106:                 double links;
   107:                 double rechts;
   108:                 int i = 0;
   109:
   110:                 while(abs(Delta)>eps){
   111:                                         ++i;
   112:                                 links = trapez(a,-lim,N/2,F1)+trapez(lim,b,N/2,F1)+trapezlim(-1,1,N/2,F1
_sing,lim);
   113:                                 rechts = trapez(a,-lim,N,F1)+trapez(lim,b,N,F1)+trapezlim(-1,1,N,F1_sing
,lim); // berechne Integral mit halber Anzahl Trapeze
   114:                                 Delta = abs(rechts-links)/links; // relativer Fehler zum vorigen Schritt
   115:                                 N = 2*N;
   116:                                 lim=lim/2;
   117:                                         }
   118:                 //cout << i << endl;
   119:                 return links;
   120:
   121:         return rechts;
   122: }
   123:
   124:
   125:  */
   126:
   127: //*************************************************************
   128: // Funktion fÃ\203Â¼r Aufgabenteil b)
   129: //*************************************************************
   130:
   131: double F2Trapez(){
   132:         double up = 2; // Obere Grenze, die jeweils um Faktor 1.5 vergrÃ¶Ã\237ert wird
   133:                 double Delta = 1;
   134:                 double N=2;
   135:                 double integral2;
   136:                 int i = 0;
   137:
   138:                     while(abs(Delta)>eps){
   139:                         ++i;
   140:                         integral2 = trapez(0,up,N,F2);
   141:                         Delta = abs(1.77245385-integral2)/1.77245385; // relativer Fehler zum "wahren" W
ert
   142:                         N = 2*N;
   143:                         up *= 1.5;
   144:                     }
   145:                     cout << "\nAnzahl Iterationen: " << i << endl;
   146:                     cout << "Wert des Integrals 2b): " << integral2 << endl;
   147:
   148:         return integral2;
   149: }
   150:
   151: //*************************************************************
   152: // Funktion fÃ\203Â¼r Aufgabenteil c)
   153: //*************************************************************
   154:
   155: double F3Trapez(){
   156:         double ob = 1;
   157:                 double Delta = 1;
   158:                 double N=2;
   159:                 double integral3;
   160:                 int i = 0;
   161:
   162:                     while(abs(Delta)>eps){
   163:                                             ++i;
   164:                           integral3 = 2 * trapez(0,ob,N,F3);
   165:                           Delta = abs(M_PI-integral3)/M_PI; // relativer Fehler zum "wahren" Wert
   166:                           N = 2*N;
   167:                           ob *= 1.5;}
```

```
168:                cout << "\nAnzahl Iterationen: " << i << endl;
169:                cout << "Wert des Integrals 2c): " << integral3 << endl;
170:
171:        return integral3;
172: }
173:
174: //*************************************
175: // main
176: //*************************************
177:
178: int main(){
179:        int a1 = -1;
180:        int b1 = 1;
181: //Aufgabenteil a)
182:        cout << "****************\nAufgabe 2a)\n****************\n" << endl;
183:        cout << scientific << setprecision(8) << Trapez(a1,b1) << endl;
184: //Aufgabenteil b)
185:        cout << "\n****************\nAufgabe 2b)\n****************" << endl;
186:        cout << scientific << setprecision(8) << F2Trapez() << endl;
187: //Aufgabenteil c)
188:        cout << "\n****************\nAufgabe 2c)\n****************" << endl;
189:        cout << scientific << setprecision(8) << F3Trapez() << endl;
190:
191:        return 0;
192: }
193:
194:
195:
```