```
// v1.1: Lösche H* Einfluss (22.09.2008)
//     23.09.2008: Erneuerung
// 16.10.2008: Kleiner Fehler behoben
// 10.02.2009: Speicheroptimierungen
// 11.02.2009: Ausgabe von u_0 Mrev vs H hinzugefuegt
// 12.02.2009: Entmagnetisierungsfaktor hinzugefügt


const double phi_0    = 2.07e-15;                       //     Flussquant
double delta_B;                                 //  Schrittweite bei Feld < 1T
double delta_Bh;                                   // Schrittweite bei Feld >= 1T

int anzahl_st;                                     //     Anzahl der Stuetzstellen fuer Gauss Legendre Integration
double *st;                                        //     Stuetzstellen bei Gauss Integration
double *we;                                        //     Gewichte bei Gauss Integration
double **phi1;                          //    Int(dz, Bz) an den Stuetzstellen, Bereich A positiv
double **phi2;                          //    Int(dz, Bz) an den Stuetzstellen, Bereich D positiv


//                      ************************************
//                      **                                    **
//                      * *             A                   * *          y
//                      *    *                           *   *           |
//                      *    *                         *     *           |
//                      *    B        *********x*********   D  *           |_____ x
//                      *         *              0            *      *
//                      *    *                               *      *
//                      * *             C                     * *
//                      **                    **
//                      ************************************

class Ha_abh                            //    Vier Groessen von Ha
{
    public:
        double B0;                      //    ausseres Feld
        double jc;                      //    kritische Stromdichte
        double B;                       //    Induktion
        double jc_rev;                  //    reversible "Stromdichte"
};

class Ha_m                              //    Zwei Groessen von Ha
{
    public:
        double B0;                      //    ausseres Feld oder Induktion
        double mag;                     //    Magnetisiereung oder Stromdichte
};


#include "gauss_pkte.h"         //    Berechnet Stuetzpunkte und Gewichte fuer Gauss Legendre Integration
#include "Jc_Ha_Bthin2.h"               //     Berechnung von B0 -> B
//#include "Jc_Ha_Bthin.h"              //     Berechnung von B0 -> B
#include "Jc_Ha_B.h"                    //    Berechnung von B0 -> B
#include "sort.h"                  //    Sortieren von Feldern der Klasse Ha_m
#include "splines.h"               //    Spline Interpolation von Feldern der Klasse Ha_m

double Max1(double a, double b)         //    Berechnet Maximum von a und b
{
    if (a > b) return a;
    return b;
}

double Min1(double a, double b)         //    Berechnet Minimum von a und b
{
    if (a < b) return a;
    return b;
}

void jc_B(char *out_datei, char *bea_datei, double a, double b, double c, double B_stern_skal, int modus, int interpol, int Stab, const int numb1,
const int numb2, char *dateirev_out, int datei2_status, double demag)
{
    //const int numb1 = 3000;                   //     Max. Anzahl an Messpunkten (bei Squidmessung)
    //const int numb2 = 6000;                   //     Max. Anzahl an Stuetzpunkte nach Spline Interpolation
    //double a, b, c;                           // Dimensionen der Probe
    double   B_stern;                           //     Streufeld bei x = 0, y = 0 und mag = 1;
    double   var, var1, d, e, dvar, mvar;
    double   d2y[numb1];                        // 2te Ableitung fuer splines
    double   d2y1[numb2], d2y2[numb2];  // 2te Ableitung fuer splines fr beide Jc(B) Zweige
    double   x_value, y_value;                  //     Datenpaar aus Splineinterpolation
    double   grenze_Min, grenze_Max;    //     Max. Induktion mit Daten von beiden Jc(B) Zweigen; Max. Induktion
```

```
    double   y_value1, y_value2;          //   y - Werte aus Splineinterpolation fuer pos. (2) und neg. (1) Jc(B) Zweig
    double    const_rev;                  //    Konstante log Ableitung der reversible Magnetisierung fuer Extrapolation
    double M_value;       // u_0 Mrev
    //char          in_datei[200];        //    Einzulesende Messdatei
    //char          out_datei[200];       //    Ausgabedatei
    char       vblah[500];
    int        number;                    //    Anzahl der Messwerte aus SQUID Messung bzw. nach spline
    int        number1, number2;          //    Anzahl der Punkte der beiden jc(B) Zweige
    int        teiler;                    // Punkt der die beiden jc(B) Zweige teilt
    int     i, j, k, start_Bp, start_Bn;

    Ha_abh  t[numb2];                     //    Ergebnisse der Rechnungen
    Ha_m    squid[numb1];                 //    Ergebnisse aus Squid - Messung
    Ha_m         z1[numb2], z2[numb2];    //    Beide Jc(B) Zweige; z1 ... neg. B0; z2 ... pos. B0
    ofstreamout2, out;
    ifstream in1, in2;

        for (i = 0; i < numb1; i++) squid[i].B0 = 1e2;          //    Initialisiere B0 mit max. Wert fuer spaezere Kontrolle
        for (i = 0; i < numb2; i++) t[i].jc_rev = 1.0e21;   // Initialisiere jc_rev mit max. Wert fuer spaetere Kontrolle

        in2.open("magout.dat", ios::in);          //    Oeffnen der Messdatei
        if (!in2)
        {
            cout << "\n\r Fehler beim Oeffnen der Messdatei";
            exit(0);
        }

        number = -1;                          //    Setze Anzahl der Messungen
        do                                    //    Einlesen der Messung
        {
            number ++;
            in2 >> squid[number].B0 >> squid[number].mag;
            if (number > numb1) cout << "FEHLER, zu viele Messpunkte"  << endl;
        }
        while (!in2.eof());                   //    Bis Ende der Datei erreicht ist
        in2.close();                          //    Schliessen der Messdatei
        while (squid[number].B0 > 0.99e2) number--;
        // cout << "number = " << number << "  squid[number].B0 = " << squid[number].B0 << "  squid[number].mag = " <<  squid[number].mag
<< endl;
        number++;

        sort_do(squid, number);                         //    Sortieren der Felder, beginne mit min. Feld

        var       = 4.0 / ( a*b*c*b * ( 1 - b/(3*a) ) );              //    Jc = mag * var

        if (interpol == 0) spline(squid, d2y, number);              //    Initialisiere Spline fuer m(B0) (Berechnung der zweiten Ableitung)

        // Berechnung und Ausgabe von Jc(H0) und u_0 Mrev(H0)

                                    out2.open("jcvsB0.dat", ios::out);                  //    Ausgabedatei
                                    if (!out2)
                                    {
                                        cout << "\n\r Fehler beim Oeffnen der Datei";
                                        exit(0);
                                    }

                                    dvar = Min1(fabs(squid[0].B0), fabs(squid[number-1].B0));
                                    x_value = 0;
                                    ipol(squid, d2y, number, x_value, y_value, interpol);
                                    out2 << x_value << " " << y_value*var << " " << 0  << endl;
                                    if (x_value < 1.0) x_value += delta_B;          //    Schrittweite fuer Ausgabe

                                    else x_value += delta_Bh;

                                    while (x_value < dvar)
                                    {
                                        ipol(squid, d2y, number, x_value, y_value1, interpol);
                                        ipol(squid, d2y, number, -x_value, y_value2, interpol);
                                        out2 << x_value << " " << 0.5*(y_value1 + y_value2)*var << " " << 0.5*(y_value1 - y_value2)/
(a*b*c)*u_0 << endl;
                                        if (x_value < 1.0) x_value += delta_B;          //    Schrittweite fuer Ausgabe

                                        else x_value += delta_Bh;
                                    }
                                    out2.close();
                                    out2.clear();
        // ENDE: Berechnung und Ausgabe von Jc(H0)

        if (Stab == 1)
```

```
{
    //    Symmetrisierung
    //cout << "ACHTUNG" << endl;
    dvar = Min1(fabs(squid[0].B0), fabs(squid[number-1].B0));
    j = 0;
    while(fabs(squid[j].B0) > dvar) j++;
    k = number-1;
    while(fabs(squid[k].B0) > dvar) k--;
    for (i=j; i<= k; i++)
    {
        x_value = squid[i].B0;
        ipol(squid, d2y, number, -x_value, y_value, interpol);
        t[i].B0 = 0.5 * (squid[i].mag - y_value);                                    //    t[] hier nur HIlfsvariable
    }
    dvar = 0.5 * (squid[j].mag - squid[k].mag);
    for (i=0; i<j; i++) t[i].B0 = dvar;
    for (i=number-1; i>k; i--) t[i].B0 = -dvar;
    for (i=0; i<number; i++) squid[i].mag -= t[i].B0;
    // ENDE:    Symmetrisierung
}

//cout << squid[-1].B0 << " " << squid[-1].mag << " " << number << endl;

x_value = squid[0].B0;                          //    Startpunkt und x-Werte fuer Splines
i    = 0;
t[i].B0 = x_value;
t[i].jc = squid[0].mag * var;
x_value += delta_B;
i++;

while (x_value <= squid[number-1].B0)          //    Interpolation von m(B0)
{
    ipol(squid, d2y, number, x_value, y_value, interpol);    //    findet y(x)
    t[i].B0  = x_value;                         //    Induktion B
    t[i].jc  = y_value * var;                   //    kritische Stromdichte jc
    x_value += delta_B;                         //    Schrittweite
    i++;
}
number = i;                                     //    Anzahl der Stuetzpunkte nach Interpolation

//cout << t[0].B << " " << t[0].B0 << " " << t[0].jc << " " << t[0].jc_rev << endl;

for (i=0; i<number; i++) t[i].B = Jc_Ha_B(t[i].B0, t[i].jc, a, b, c, modus);   // jc(B0) -> jc(B) nur positive B - Werte!

            // Testausgabe
                out.open("jcout1.dat", ios::out);
                if (!out)
                    {
                    cout << "\n\r Fehler beim Erstellen der Datei ";
                    exit(0);
                    }
                for (i=0; i<number; i++) out << t[i].B << " " << t[i].B0 << " " << t[i].jc <<endl;
                out.close();
                out.clear();
                                                                                    // Ende: Testausgabe

 //cout << t[0].B << " " << t[0].B0 << " " << t[0].jc << " " << t[0].jc_rev << endl;

teiler = 0;
while (t[teiler +1].B < t[teiler].B) teiler++;          //    Suche Grenze zwischen den beiden B - Zweigen
//cout << "\n teiler = " << teiler << " t[teiler].B = " << t[teiler].B << " t[teiler].B0 = " << t[teiler].B0 << endl;

start_Bn = teiler;                              //    Startindex der Jc(B) Werte aus negativem B0 Zweig; i = 0...start_Bn
while (t[start_Bn-1].jc > t[start_Bn].jc) start_Bn--;
//if  (t[start_Bn].jc > t[start_Bn+1].jc) start_Bn--;
//cout << "\n start_Bn = " << start_Bn << " t[start_Bn].B = " << t[start_Bn].B << endl;


start_Bp = number-1;                            //    Startindex der Jc(B) Werte aus positiven B0 Zweig; i = start_Bp...number-1
//cout << "\n t[start_Bp].B = " << t[start_Bp].B << endl;
//cout << "\n t[start_Bp-1].B = " << t[start_Bp-1].B << endl;
//cout << "\n t[start_Bn].B = " << t[start_Bn].B << endl;
//while (t[start_Bp-1].B > t[start_Bn].B) start_Bp--;
while (t[start_Bp-1].B < t[start_Bp].B)  start_Bp--;
while (t[start_Bp+1].jc > t[start_Bp].jc)       start_Bp++;
//if (t[start_Bp].jc > t[start_Bp-1].jc) start_Bp++;
//cout << "\n start_Bp = " << start_Bp << "  t[start_Bp].B = " << t[start_Bp].B << endl;
//cout << "\n start_Bn = " << start_Bn << "  t[start_Bn].B = " << t[start_Bn].B << endl;
```

```
    if (t[start_Bp].B < t[start_Bn].B) {while (t[start_Bp+1].B < t[start_Bn].B) start_Bp++;}
    if (t[start_Bn].B < t[start_Bp].B) {while (t[start_Bn-1].B < t[start_Bp].B) start_Bn--;}

    //cout << "\n start_Bp = " << start_Bp << "  t[start_Bp].B = " << t[start_Bp].B << endl;
    //cout << "\n start_Bn = " << start_Bn << "  t[start_Bn].B = " << t[start_Bn].B << endl;


    number1 = start_Bn+1;
    number2 = number - start_Bp;       //    Anzahl der Jc(B) Werte aus positivem B0 Zweig
     // cout << "number1 = " << number1 << "  number2 = " << number2 << endl;


    for (i = 0; i <= start_Bn; i++)                 //    Initialisiere neue Klasse Ha_m fuer -B0 Zweig
    {
        z1[i].B0  = t[i].B;                         //    z1.B0  = B   !!!
        z1[i].mag = t[i].jc;                            // z1.mag = jc  !!!
    }
    for (i = number - 1; i >= start_Bp; i--)   //    Initialisiere neue Klasse Ha_m fuer +B0 Zweig
    {
        z2[number - 1 -i].B0 = t[i].B;          //    z2.B0  = B   !!!
        z2[number - 1 -i].mag = t[i].jc;        // z2.mag = jc  !!!
    }

    sort_do(z1, number1);                           //    sortiere -B0 Zweig, kleinstes B steht zu Beginn
    sort_do(z2, number2);                           //    sortiere +B0 Zweig, kleinstes B steht zu Beginn



    if (interpol == 0) spline(z1, d2y1, number1);                   //    Initialisiere Spline fuer beide Zweige
    if (interpol == 0) spline(z2, d2y2, number2);
                                                                    //    Berechnung der reversiblen Magnetisierung mag_rev(B) und entsprechende
                                                                    //    Korrektur von mag(B0) - eigentlich Jc_rev(B) und jc(B0)


    var1      = Max1(z1[0].B0, z2[0].B0);
    var  = Min1(z1[number1-1].B0, z2[number2-1].B0);


// cout << "ok1 " << endl;

    // Berechne Rev-Jc


    for (i=0; i<number; i++) t[i].jc_rev = 1e99;        // Markiere Rev_Jc

    // Negativer Zweig:
    for (i = 0; i <= start_Bn; i++)
    {
        if (t[i].B >= var1 && t[i].B <= var)
        {
            ipol(z2, d2y2, number2, t[i].B, y_value, interpol);
            t[i].jc_rev = 0.5 * (t[i].jc - y_value);
        }
    }

// cout << "ok1a " << endl;

    i = 0;                                  // Extrapoliere bei großen Feldern
    while (t[i].jc_rev > 1e98 && i < number) i++;
    if (i>0)
    {
        if ((start_Bn - i) > 3) dvar = (t[i].jc_rev + t[i+1].jc_rev + t[i+2].jc_rev) / 3.0;
        else dvar = t[i].jc_rev;
        for (j=0; j<i; j++) t[j].jc_rev = dvar;                 // Konstantes Jc-Rev
    }
    i = teiler;                                 // Extrapoliere bei kleinen Feldern
    while (t[i].jc_rev > 1e98 && i <number) i--;
    if (i < teiler)
    {
        //dvar = t[i].jc_rev / t[i].B;      //    Steigung
        //for (j=teiler; j>i; j--) t[j].jc_rev = t[j].B * dvar;
        dvar = t[i].jc_rev;             //    Steigung
        for (j=teiler; j>i; j--) t[j].jc_rev = dvar;
    }
        t[teiler].jc_rev = 1e99;
    // Ende: Negativer Zweig:

// cout << "ok1b " << endl;
```

```
    // Positiver  Zweig:
    for (i = start_Bp; i < number; i++)
    {
        if (t[i].B >= var1 && t[i].B <= var)
        {
            ipol(z1, d2y1, number1, t[i].B, y_value, interpol);
            t[i].jc_rev = 0.5 * (t[i].jc - y_value);
        }
    }
    // cout << "ok1c " << endl;
    i = number-1;                                   // Extrapoliere bei großen Feldern
    while (t[i].jc_rev > 1e98 && i < number) i--;
    // cout << "ok1ca " << endl;
    if (i < number-1)
    {
        if ((start_Bp - i) > 3) dvar = (t[i].jc_rev + t[i-1].jc_rev + t[i-2].jc_rev) / 3.0;
        else dvar = t[i].jc_rev;
        // cout << "ok1cb " << endl;
        // cout << number-1 << "   " << i << "  " << dvar << endl;
        for (j=number-1; j>i; j--) t[j].jc_rev = dvar;          // Konstantes Jc-Rev
    }
    // cout << "ok1d " << endl;
    i = teiler;                                 // Extrapoliere bei kleinen Feldern
    while (t[i].jc_rev > 1e98 && i <number) i++;
    if (i > teiler)
    {
        dvar = t[i].jc_rev / t[i].B;        //    Steigung
        for (j=teiler; j<i; j++) t[j].jc_rev = t[j].B * dvar;
    }
    // cout << "ok3 " << endl;
    // Ende: Positiver Zweig:
    t[teiler].jc_rev = 0.5*(t[teiler+1].jc_rev + t[teiler-1].jc_rev);

    // Ende: Berechne Rev-Jc

// cout << "ok4 " << endl;
     //cout << t[0].B << " " << t[0].B0 << " " << t[0].jc << " " << t[0].jc_rev << endl;

    for (i=0; i<number; i++) t[i].jc -=  t[i].jc_rev;        // Erneuere Jc
    for (i=0; i<number; i++) t[i].B = Jc_Ha_B(t[i].B0, t[i].jc, a, b, c, modus);          // Berechne erneut B

     //cout << t[0].B << " " << t[0].B0 << " " << t[0].jc << " " << t[0].jc_rev << endl;

// cout << "ok5 " << endl;

                    // Testausgabe
                    out.open("jcout2.dat", ios::out);
                    if (!out)
                    {
                        cout << "\n\r Fehler beim Erstellen der Datei ";
                        exit(0);
                    }
                    for (i=0; i<number; i++) out << t[i].B << " " << t[i].B0 << " " << t[i].jc << " " << t[i].jc_rev << endl;
                    out.close();
                    out.clear();
                    // Ende: Testausgabe




//    Ausgabe von u_0 Mrev vs H

    if (datei2_status == 1)
    {
        mvar          = 4.0 / ( b * ( 1 - b/(3*a) ) );
        mvar       = -u_0 / mvar;       //    u_0 Mrev = Jc  *mvar
        out2.open(dateirev_out, ios::out);                          //    Ausgabedatei
        if (!out2)
        {
            cout << "\n\r Fehler beim Oeffnen der Datei";
            exit(0);
        }

        ipol(z1, d2y1, number1, var1, y_value1, interpol);
        ipol(z2, d2y2, number2, var1, y_value2, interpol);
        M_value      = 0.5*(y_value1 - y_value2) * mvar;            // u_0 Mrev
```

```
        out2 << var1 << " " << var1 - M_value*demag << " " << var1 - M_value*demag + M_value << " " <<  M_value << endl;

        x_value = 0;
        while (x_value < var1)
        {
            if (x_value < 1.0) x_value += delta_B;                  //    Schrittweite fuer Ausgabe
            else x_value += delta_Bh;
        }
        while (x_value < var)
        {
            ipol(z1, d2y1, number1, x_value, y_value1, interpol);
            ipol(z2, d2y2, number2, x_value, y_value2, interpol);
            M_value      = 0.5*(y_value1 - y_value2) * mvar;                    // u_0 Mrev
            out2 << x_value << " " << x_value - M_value*demag << " " << x_value - M_value*demag + M_value << " " <<  M_value << endl;
            if (x_value < 1.0) x_value += delta_B;                  //    Schrittweite fuer Ausgabe
            else x_value += delta_Bh;
        }
        out2.close();
        out2.clear();
    }
        // Ende   Ausgabe von u_0 Mrev vs H




    // Erneute Teilung

    teiler = 0;
    while (t[teiler +1].B < t[teiler].B) teiler++;              //    Suche Grenze zwischen den beiden B - Zweigen
    //cout << "teiler = " << teiler << "t[teiler].B = " << t[teiler].B << endl;

    start_Bn = teiler-1;                                        //    Startindex der Jc(B) Werte aus negativem B0 Zweig; i = 0...start_Bn
    while (t[start_Bn-1].jc > t[start_Bn].jc) start_Bn--;
    //if  (t[start_Bn].jc > t[start_Bn+1].jc) start_Bn--;
    // cout << "start_Bn = " << start_Bn << "t[start_Bn].B = " << t[start_Bn].B << endl;

    start_Bp = number-1;
    //start_Bp = teiler;                                        //    Startindex der Jc(B) Werte aus negativem B0 Zweig; i = start_Bp...number-1

    while (t[start_Bp-1].B < t[start_Bp].B)  start_Bp--;
    if (start_Bp == teiler) start_Bp++;
    while (t[start_Bp+1].jc > t[start_Bp].jc)       start_Bp++;
    //if (t[start_Bp].jc > t[start_Bp-1].jc) start_Bp++;
    //if (start_Bn == teiler) start_Bn--;

//  cout << "\n start_Bp = " << start_Bp << "  t[start_Bp].B = " << t[start_Bp].B << endl;
//  cout << "\n start_Bn = " << start_Bn << "  t[start_Bn].B = " << t[start_Bn].B << endl;

    if (t[start_Bp].B < t[start_Bn].B) {while (t[start_Bp+1].B < t[start_Bn].B) start_Bp++;}
    if (t[start_Bn].B < t[start_Bp].B) {while (t[start_Bn-1].B < t[start_Bp].B) start_Bn--;}

//  cout << "\n start_Bp = " << start_Bp << "  t[start_Bp].B = " << t[start_Bp].B << endl;
//  cout << "\n start_Bn = " << start_Bn << "  t[start_Bn].B = " << t[start_Bn].B << endl;


    //while (t[start_Bp+1].jc > t[start_Bp].jc) start_Bp++;
    //if (t[start_Bp].jc > t[start_Bp-1].jc) start_Bp++;
    // cout << "start_Bp = " << start_Bp << "t[start_Bp].B = " << t[start_Bp].B << endl;
    number1 = start_Bn+1;
    number2 = number - start_Bp;       //    Anzahl der Jc(B) Werte aus positivem B0 Zweig
    // cout << "number1 = " << number1 << "  number2 = " << number2 << endl;


    for (i = 0; i <= start_Bn; i++)                 //    Initialisiere neue Klasse Ha_m fuer -B0 Zweig
    {
        z1[i].B0  = t[i].B;                         //    z1.B0  = B   !!!
        z1[i].mag = t[i].jc;                            // z1.mag = jc  !!!
    }
    for (i = number - 1; i >= start_Bp; i--)   //    Initialisiere neue Klasse Ha_m fuer +B0 Zweig
    {
        z2[number - 1 -i].B0 = t[i].B;          //    z2.B0  = B   !!!
        z2[number - 1 -i].mag = t[i].jc;        // z2.mag = jc  !!!
    }

    sort_do(z1, number1);                               //    sortiere -B0 Zweig, kleinstes B steht zu Beginn
    sort_do(z2, number2);                               //    sortiere +B0 Zweig, kleinstes B steht zu Beginn
```

```
        if (interpol == 0) spline(z1, d2y1, number1);                   //    Initialisiere Spline fuer beide Zweige
        if (interpol == 0) spline(z2, d2y2, number2);


        var1       = Max1(z1[0].B0, z2[0].B0);
        var  = Min1(z1[number1-1].B0, z2[number2-1].B0);


                                                                                // Testausgabe
                                                                                        out.open("jcbranch.dat", ios::out);
                                                                                        if (!out)
                                                                                        {
                                                                                             cout << "\n\r Fehler beim Erstellen der Datei ";
                                                                                             exit(0);
                                                                                        }
                                                                                        for (i=0; i<number1; i++) out << z1[i].B0 << "   " <<
z1[i].mag <<endl;
                                                                                        out << endl;
                                                                                        for (i=0; i<number2; i++) out << z2[i].B0 << "   " <<
z2[i].mag <<endl;
                                                                                        out.close();
                                                                                        out.clear();

                                                                                        // Ende: Testausgabe

        //    Ausgabe von Jc vs B
        out2.open(out_datei, ios::out);                         //    Ausgabedatei
        if (!out2)
        {
             cout << "\n\r Fehler beim Oeffnen der Datei";
             exit(0);
        }

        if (var1 == z1[0].B0) y_value1 = z1[0].mag;
        else ipol(z1, d2y1, number1, var1, y_value1, interpol);
        if (var1 == z2[0].B0) y_value2 = z2[0].mag;
        else ipol(z2, d2y2, number2, var1, y_value2, interpol);
        out2 << var1 << " " << 0.5*(y_value1 + y_value2) << endl;
        x_value = 0;
        while (x_value < var1)
        {
             if (x_value < 1.0) x_value += delta_B;                 //    Schrittweite fuer Ausgabe
             else x_value += delta_Bh;
        }

        while (x_value < var)
        {
             ipol(z1, d2y1, number1, x_value, y_value1, interpol);
             ipol(z2, d2y2, number2, x_value, y_value2, interpol);
             out2 <<x_value << " " << 0.5*(y_value1 + y_value2) << endl;
             if (x_value < 1.0) x_value += delta_B;                 //    Schrittweite fuer Ausgabe
             else x_value += delta_Bh;
        }
        dvar = 0.5 * (y_value1 - y_value2);
        while (x_value < z1[number1-1].B0)
        {
             ipol(z1, d2y1, number1, x_value, y_value1, interpol);
             out2 << x_value << " " << y_value1 - dvar << endl;
             if (x_value < 1.0) x_value += delta_B;                 //    Schrittweite fuer Ausgabe
             else x_value += delta_Bh;
        }
        while (x_value < z2[number2-1].B0)
        {
             ipol(z2, d2y2, number2, x_value, y_value2, interpol);
             out2 << x_value << " " << y_value2 + dvar << endl;
             if (x_value < 1.0) x_value += delta_B;                 //    Schrittweite fuer Ausgabe
             else x_value += delta_Bh;
        }
        out2.close();
        out2.clear();
             // Ende   Ausgabe von Jc vs B



}
```