HMNull

Version: 12.05.2016

Numerische Verfahren zur Nullstellenbestimmung in C und C++

Tilman Küpper (tilman.kuepper@hm.edu)

Inhalt

1. Einleitung

In der Funktionsbibliothek HMNull sind die folgenden drei Verfahren zur numerischen Nullstellenbestimmung implementiert:

- das Bisektionsverfahren (auch Intervallhalbierungsverfahren),
- das Newtonverfahren (auch Newton-Raphson-Verfahren),
- das Sekantenverfahren.

HMNull kann sowohl in C- als auch in C++-Projekten verwendet werden. Unabhängig von der verwendeten Programmiersprache (C oder C++) sind die beiden Dateien **hmnull.c** und **hmnull.h** dem Projekt hinzuzufügen.

In der folgenden Abbildung ist die Entwicklungsumgebung „Qt Creator“ mit einem geöffneten C++-Projekt dargestellt. Die Seitenleiste am linken Fensterrand zeigt eine Übersicht über das aktuell geöffnete Projekt, hier sind die genannten Dateien aufgelistet.

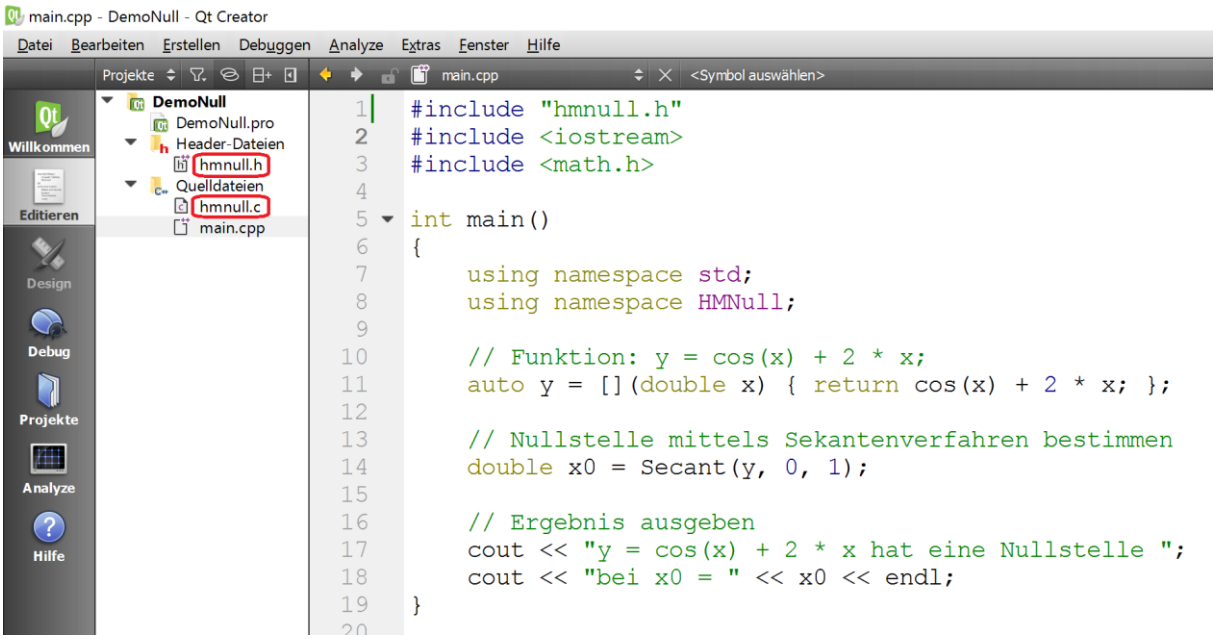

Abbildung 1 – Qt Creator mit einem geöffneten C++-Projekt

Hinweis: Zur Berechnung reeller und komplexer Nullstellen von Polynomen ist das Weierstraß-Durand-Kerner-Verfahren gut geeignet. Dieses ist allerdings nicht im Rahmen von HMNull sondern in der Funktions- und Klassenbibliothek HMMatrix implementiert. Die dortigen Funktionsnamen lauten für C-Projekte: v_croots(), v_croots2(), v_droots() und v_droots2(), bzw. für C++-Projekte: Vector::Roots().

2. Programmierung in C

2.1. Beispiel: Bisektionsverfahren

```c
/* ------------------------------------------------------------------------ */
/* HMNULL: Berechnung einer Nullstelle mit dem Bisektionsverfahren in C     */
/* ------------------------------------------------------------------------ */
#include "hmnull.h"
#include <stdio.h>
#include <math.h>

/* f(x) = sin(x) * cos(x) */
double fx(double x, void *pdata)
{
    return sin(x) * cos(x);
}

/* f(x) hat bei pi/2 eine Nullstelle */
int main(void)
{
    double x0 = bisect(fx, 1.0, 2.0);
    printf("Nullstelle bei x0 = %f\n", x0);  /* 1.5708 */
    return 0;
}
```

2.2. Beispiel: Newtonverfahren

```c
/* ------------------------------------------------------------------------ */
/* HMNull: Berechnung einer Nullstelle mit dem Newtonverfahren in C;        */
/* zusätzlich zur Funktion f(x) muss deren Ableitung übergeben werden.      */
/* ------------------------------------------------------------------------ */
#include "hmnull.h"
#include <stdio.h>
#include <math.h>

/* f(x) = x^2 - 1 */
double fx(double x, void *pdata)
{
    return x * x - 1.0;
}

/* f'(x) = 2x */
double df_dx(double x, void *pdata)
{
    return 2 * x;
}

/* f(x) hat Nullstellen bei x0 = +/-1.0 */
int main(void)
{
    double x0 = newton(fx, df_dx, 1.5);
    printf("Nullstelle bei x0 = %f\n", x0);  /* 1.000 */

    x0 = newton(fx, df_dx, -2.0);
    printf("Nullstelle bei x0 = %f\n", x0);  /* -1.000 */
    return 0;
}
```

2.3. Beispiel: Sekantenverfahren

```c
/* ------------------------------------------------------------------------ */
/* HMNull: Berechnung einer Nullstelle mit dem Sekantenverfahren in C       */
/* ------------------------------------------------------------------------ */
#include "hmnull.h"
#include <stdio.h>
#include <math.h>

/* f(x) = sin(x) * cos(x) */
double fx(double x, void *pdata)
{
    return sin(x) * cos(x);
}

/* f(x) hat bei pi/2 eine Nullstelle */
int main(void)
{
    double x0 = secant(fx, 1.0, 2.0);
    printf("Nullstelle bei x0 = %f\n", x0);  /* 1.5708 */
    return 0;
}
```

2.4. Beispiel: Genauigkeit und Iterationsanzahl vorgeben

```c
/* ------------------------------------------------------------------------ */
/* Auf Wunsch können gewünschte Genauigkeit (epsilon) und maximal zulässige */
/* Iterationensanzahl (max_iter) vorgegeben werden. Ebenso ist es möglich,  */
/* nach Berechnungsende einen Fehlercode zu erhalten. Über "pdata" werden   */
/* Daten vom Hauptprogramm an die Implementierung von f(x) weitergeleitet.  */
/* ------------------------------------------------------------------------ */
#include "hmnull.h"
#include <assert.h>
#include <stdio.h>
#include <math.h>

/* f(x) = sin(x) * cos(x) */
double fx(double x, void *pdata)
{
    assert(pdata == (void*)12345);  /* Daten vom Hauptprogramm... */
    return sin(x) * cos(x);
}

/* f(x) hat bei pi/2 eine Nullstelle */
int main(void)
{
    size_t max_iter = 20;             int err = 0;
    double epsilon = 1e-6;            void *my_data = (void*)12345;

    double x0 = secant_ex(fx, 1.0, 2.0, epsilon, max_iter, &err, my_data);
    if(err == HMNULL_OK)
        printf("Nullstelle bei x0 = %f\n", x0);  // 1.5708
    else
        printf("%s\n", hmnull_errmsg[err]);

    max_iter = 2;  /* Nur zwei Iterationen reichen nicht aus! */
    x0 = secant_ex(fx, 1.0, 2.0, epsilon, max_iter, &err, my_data);
    if(err == HMNULL_OK)
        printf("Nullstelle bei x0 = %f\n", x0);
    else
        printf("%s\n", hmnull_errmsg[err]);  /* "Iteration failed" */

    return 0;
}
```

2.5. Funktionsübersicht

Rückgabewerte bzw. Fehlermeldungen von Funktionen:

```
#define HMNULL_OK          0   /* Funktion wurde erfolgreich beendet          */
#define HMNULL_ERR_ITFAIL  1   /* Berechnungsverfahren konvergiert nicht      */
#define HMNULL_ERR_PARAM   2   /* Ungültige Parameter beim Funktionsaufruf    */
#define HMNULL_ERR_NAN     3   /* Überlauf/NaN während der Berechnung         */
```

Defaultwerte für die zu erreichende Genauigkeit und die maximal zulässige Iterationsanzahl:

```
#define HMNULL_EPSILON 1.0e-10
#define HMNULL_MAXITER 10000UL
```

- ```
  double bisect_ex(hmnull_fn fn, double x1, double x2,
      double epsilon, size_t max_iter, int *errorcode, void *pdata);
  ```

  Numerische Bestimmung einer Nullstelle mittels Bisektionsverfahren: Als Parameter "fn" ist die Funktion zu übergeben, deren Nullstelle bestimmt werden soll (vergl. "typedef hmnull_fn" in "hmnull.h"). Die Nullstelle muss zwischen x1...x2 liegen, fn(x1) und fn(x2) müssen unterschiedliche Vorzeichen haben. Das Verfahren wird beendet, falls eine Lösung mit der Genauigkeit "epsilon" ermittelt oder nach "max_iter" Iterationen keine Lösung gefunden wurde. Über "pdata" kann ein beliebiger Zeiger an die Implementierung von "fn" weitergeleitet werden.

  Über "errorcode" wird ein Fehlercode zurückgegeben, mögliche Codes sind: HMNULL_ERR_ITFAIL, HMNULL_ERR_PARAM, HMNULL_ERR_NAN und HMNULL_OK. (Es ist zulässig, im Parameter "errcode" NULL zu übergeben, falls kein Fehlercode benötigt wird.) Die berechnete Nullstelle wird als Rückgabewert zurückgegeben.

- ```
  double bisect(hmnull_fn fn, double x1, double x2);
  ```

  Numerische Bestimmung einer Nullstelle mittels Bisektionsverfahren: Siehe Dokumentation von "bisect_ex" für Details zum Funktionsaufruf. Im Gegensatz zu "bisect_ex" werden hier für die Parameter "epsilon", "max_iter", "errcode" und "pdata" folgende Standardwerte übergeben: HMNULL_EPSILON, HMNULL_MAXITER, NULL und NULL.

- ```
  double newton_ex(hmnull_fn fn, hmnull_fn deriv, double x0,
      double epsilon, size_t max_iter, int *errorcode, void *pdata);
  ```

  Numerische Bestimmung einer Nullstelle mittels Newtonverfahren: Als Parameter "fn" ist die Funktion zu übergeben, deren Nullstelle bestimmt werden soll (vergl. "typedef hmnull_fn" in "hmnull.h"). In "deriv" ist die Ableitung von "fn" zu übergeben. Der Algorithmus beginnt mit dem Startwert "x0". Er wird beendet, falls eine Lösung mit der Genauigkeit "epsilon" ermittelt oder nach "max_iter" Iterationen noch keine Lösung gefunden wurde. Über "pdata" kann ein Zeiger an die Funktionen "fn" und "deriv" weitergeleitet werden.

  Über "errorcode" kann ein Fehlercode an den Aufrufer zurückgegeben werden, mögliche Codes: HMNULL_ERR_ITFAIL, HMNULL_ERR_NAN und HMNULL_OK. (Es ist zulässig, im Parameter "errcode" NULL zu übergeben, falls kein Fehlercode benötigt wird.) Die berechnete Nullstelle wird als Rückgabewert zurückgegeben.

- ```
  double newton(hmnull_fn fn, hmnull_fn deriv, double x0);
  ```

  Numerische Bestimmung einer Nullstelle mittels Newtonverfahren: Siehe Dokumentation von "newton_ex" für Details zum Funktionsaufruf. Im Gegensatz zu "newton_ex" werden hier für die Parameter "epsilon", "max_iter", "errcode" und "pdata" folgende Standardwerte übergeben: HMNULL_EPSILON, HMNULL_MAXITER, NULL und NULL.

- ```
  double secant_ex(hmnull_fn fn, double x1, double x2,
      double epsilon, size_t max_iter, int *errorcode, void *pdata);
  ```

  Numerische Bestimmung einer Nullstelle mittels Sekantenverfahren: Als Parameter "fn" ist die Funktion zu übergeben, deren Nullstelle bestimmt werden soll (vergl. "typedef hmnull_fn" in "hmnull.h"). Die Nullstelle sollte dabei möglichst in der Nähe des Intervalls x1...x2 liegen.

  Das Verfahren wird beendet, falls eine Lösung mit der Genauigkeit "epsilon" ermittelt oder nach "max_iter" Iterationen keine Lösung gefunden wurde. Über "pdata" kann ein beliebiger Zeiger an "fn" weitergeleitet werden.

  Über den Zeiger "errorcode" kann ein Fehlercode zurückgegeben werden, mögliche Codes sind: HMNULL_ERR_ITFAIL, HMNULL_ERR_NAN und HMNULL_OK. (Als "errcode" kann NULL übergeben werden, falls kein Fehlercode benötigt wird.) Die berechnete Nullstelle wird als Rückgabewert zurückgegeben.

- ```
  double secant(hmnull_fn fn, double x1, double x2);
  ```

  Numerische Bestimmung einer Nullstelle mittels Sekantenverfahren: Siehe Dokumentation von "secant_ex" für Details zum Funktionsaufruf. Im Gegensatz zu "secant_ex" werden hier für die Parameter "epsilon", "max_iter", "errcode" und "pdata" folgende Standardwerte übergeben: HMNULL_EPSILON, HMNULL_MAXITER, NULL und NULL.

3. Programmierung in C++

3.1. Beispiel: Bisektionsverfahren

```
// ----------------------------------------------------------------------------
// HMNull: Berechnung einer Nullstelle mit dem Bisektionsverfahren in C++;
// f(x) wird in diesem Beispiel als Funktionszeiger übergeben.
// ----------------------------------------------------------------------------
#include "hmnull.h"
#include <iostream>

using namespace std;
using namespace HMNull;

// f(x) = sin(x) * cos(x)
double fx(double x)
{
    return sin(x) * cos(x);
}

// f(x) hat bei pi/2 eine Nullstelle
int main()
{
    auto x0 = Bisect(fx, 1.0, 2.0);
    cout << "Nullstelle bei x0 = " << x0 << endl;  // 1.5708
}
```

3.2. Beispiel: Newtonverfahren

```
// ----------------------------------------------------------------------------
// HMNull: Berechnung einer Nullstelle mit dem Newtonverfahren in C++;
// f(x) und f'(x) werden in diesem Beispiel als Lambda-Ausdrücke übergeben.
// ----------------------------------------------------------------------------
#include "hmnull.h"
#include <iostream>

using namespace std;
using namespace HMNull;

// f(x) = x^2 - 1 hat Nullstellen bei x0 = +/-1.0
int main()
{
    auto fx    = [](double x) { return x * x - 1.0; };
    auto df_dx = [](double x) { return 2 * x; };
    auto x0 = Newton(fx, df_dx, 3.0);
    cout << "Nullstelle bei x0 = " << x0 << endl;  // 1.0
}
```

3.3. Beispiel: Sekantenverfahren

```
// ----------------------------------------------------------------------------
// HMNull: Berechnung einer Nullstelle mit dem Sekantenverfahren in C++;
// f(x) wird in diesem Beispiel als Funktionsobjekt ("Functor") übergeben.
// ----------------------------------------------------------------------------
#include "hmnull.h"
#include <iostream>

using namespace std;
using namespace HMNull;

// Funktionsobjekt zur Berechnung von f(x) = x^2 + offset
class MyFunction
{
    double offset_;

public:
    MyFunction(double offset) : offset_(offset) { }
    double operator()(double x) { return x * x + offset_; }
};

// f(x) = x^2 - 1 hat Nullstellen bei x0 = +/-1.0
int main()
{
    MyFunction fx(-1.0);
    auto x0 = Secant(fx, 5.0, 10.0);
    cout << "Nullstelle bei x0 = " << x0 << endl;  // 1.0
}
```

3.4. Beispiel: Abfragen von Berechnungsfehlern

```
// ----------------------------------------------------------------------------
// HMNull: Bei der Programmierung in C++ werden Ausnahmefehler ausgelöst,
// falls es bei der Nullstellenbestimmung zu Fehlern kommt.
// ----------------------------------------------------------------------------
#include "hmnull.h"
#include <iostream>

using namespace std;
using namespace HMNull;

// Beispiel: Beim Bisektionsverfahren muss im Intervall, das der
// Anwender angibt, eine ungerade Anzahl von Nullstellen liegen.
int main()
{
    auto y = [](double x) { return sin(x); };

    try
    {
        // Im angegebenen Intervall liegen aber zwei Nullstellen!
        double x0 = Bisect(y, -0.1, 3.2);
        cout << "Nullstelle bei x0 = " << x0 << endl;
    }
    catch(const exception &err)
    {
        // Es wird ausgegeben: "Invalid parameter"
        cout << "Fehler: " << err.what() << endl;
    }
}
```

3.5. Funktionsübersicht

Die folgenden C++-Funktionen sind im Namensraum "HMNull" definiert.

- ```
  template<typename FunctT> double Bisect(
      FunctT funct, double x1, double x2,
      double epsilon = HMNULL_EPSILON, size_t maxiter = HMNULL_MAXITER);
  ```

  Nullstellenbestimmung mit dem Bisektionsverfahren: Die Funktion f(x) ist als Funktionsobjekt (bzw. Lambda-Ausdruck, Funktionszeiger o. ä.) zu übergeben. Zusätzlich ist ein Intervall x1...x2 anzugeben, in dem die zu suchende Nullstelle (oder eine größere, ungerade Anzahl von Nullstellen) liegen muss. Optional können die gewünschte Genauigkeit sowie die maximal zulässige Iterationsanzahl angegeben werden.

- ```
  template<typename FunctT, typename DerivT> double Newton(
      FunctT funct, DerivT deriv, double x0,
      double epsilon = HMNULL_EPSILON, size_t maxiter = HMNULL_MAXITER);
  ```

  Nullstellenbestimmung mit dem Newtonverfahren: Die Funktion f(x) und deren Ableitung sind als Funktionsobjekte (bzw. Lambda-Ausdrücke, Funktionszeiger o. ä.) zu übergeben. Zusätzlich ist der Startwert x0 für das Iterationsverfahren anzugeben. Optional können die zu erreichende Genauigkeit sowie die maximal zulässige Iterationsanzahl angegeben werden.

- ```
  template<typename FunctT> double Secant(
      FunctT funct, double x1, double x2,
      double epsilon = HMNULL_EPSILON, size_t maxiter = HMNULL_MAXITER);
  ```

  Nullstellenbestimmung mit dem Sekantenverfahren: Die Funktion f(x) ist als Funktionsobjekt (bzw. Lambda-Ausdruck, Funktionszeiger o. ä.) zu übergeben. Zusätzlich ist ein Intervall x1...x2 anzugeben, in dessen Nähe die gesuchte Nullstelle liegen sollte. Optional können die gewünschte Genauigkeit sowie die maximal zulässige Anzahl von Iterationen angegeben werden.
  ```

Anhang A: Kontakt, Lizenz

Tilman Küpper
tilman.kuepper@hm.edu

Hochschule München
Fakultät für Maschinenbau, Fahrzeugtechnik, Flugzeugtechnik
Dachauer Straße 98 b
D-80335 München

http://kuepper.userweb.mwn.de/

Die Funktions- und Klassenbibliothek HMNull darf unter Beachtung der folgenden Lizenzbedingungen frei verwendet und weitergegeben werden: