HMDGL

Version: 2017-06-12

Funktionen zur numerischen Lösung von Differentialgleichungen

Tilman Küpper (tilman.kuepper@hm.edu)

Inhalt

1. Einleitung

HMDGL ermöglicht die numerische Lösung von gewöhnlichen Differentialgleichungen in den Programmiersprachen C und C++. Die Differentialgleichung muss in der Form u' = f(u, t) vorliegen, Differentialgleichungen n-ter Ordnung sind daher zunächst in ein System von n Differentialgleichungen erster Ordnung umzuwandeln, u' und u sind in diesem Fall Vektoren mit jeweils n Elementen.

Zur Programmierung in C sind die beiden Dateien **hmdgl.c** und **hmdgl.h** zum aktuellen Projekt hinzuzufügen, zur Programmierung in C++ muss nur die eine Datei **hmdgl.hpp** zum aktuellen Projekt hinzugefügt werden. Die folgende Abbildung zeigt die Entwicklungsumgebung Microsoft Visual C++ 2010 mit einem geöffneten Projekt. Auf der linken Seite ist der Projektmappen-Explorer (engl. Solution Explorer) sichtbar, in dem die genannten Dateien aufgelistet sind.

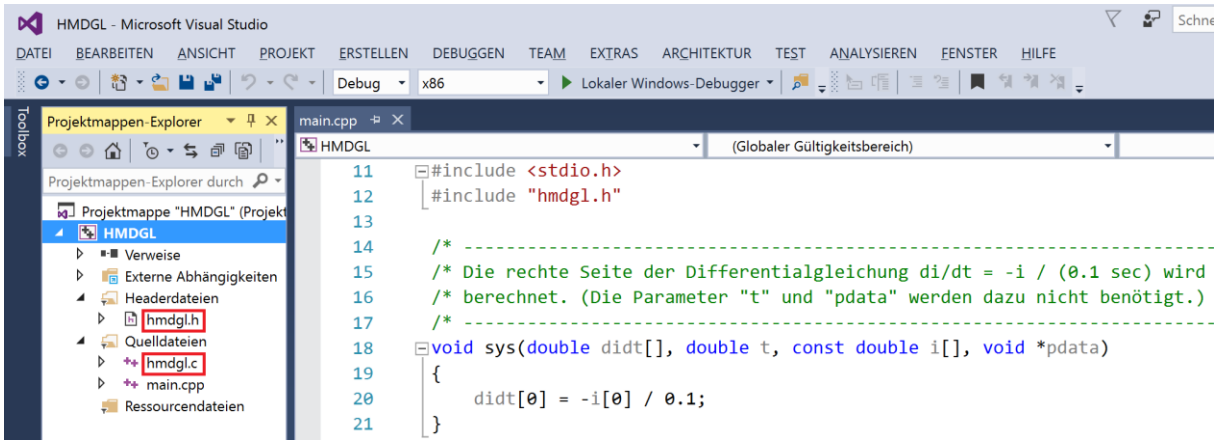

Abbildung 1 – Benutzeroberfläche von Microsoft Visual C++ 2015

2. Beispiele

2.1. Differentialgleichung erster Ordnung

```c
/* ------------------------------------------------------------------------ */
/* Ein 100 µF-Kondensator wird über einen 1000 Ω-Widerstand an eine         */
/* Spannung von 10 V angeschlossen. Für den Aufladestrom i(t) gilt:         */
/*                                                                          */
/* di/dt = -i / (0.1 sec)       Startwert: i(0 sec) = 0.01 A                */
/*                                                                          */
/* Es wird der Stromverlauf i(t) im Zeitintervall 0...1 sec ermittelt,      */
/* die Ausgabe der berechneten Werte erfolgt auf dem Bildschirm.            */
/* ------------------------------------------------------------------------ */
#include <stdio.h>
#include "hmdgl.h"

/* ------------------------------------------------------------------------ */
/* Die rechte Seite der Differentialgleichung di/dt = -i / (0.1 sec) wird   */
/* berechnet, die Parameter "t" und "pdata" werden dazu nicht benötigt.     */
/* ------------------------------------------------------------------------ */
void sys(double didt[], double t, const double i[], void *pdata)
{
    didt[0] = -i[0] / 0.1;
}

/* ------------------------------------------------------------------------ */
/* Nach jedem Integrationsschritt wird die Funktion <step> aufgerufen.      */
/* ------------------------------------------------------------------------ */
void step(double t, const double i[], void *pdata)
{
    /* Aktuelle "Simulationszeit" und Stromstärke ausgeben */
    printf("t = %6.3f sec, i = %6.4f mA\n", t, 1000.0 * i[0]);
}

/* ------------------------------------------------------------------------ */
/* Die Funktion <init> legt die Anfangswerte fest, hier: i(0 sec) = 0.01 A  */
/* ------------------------------------------------------------------------ */
void init(double i[], void *pdata)
{
    i[0] = 0.01;
}

/* ------------------------------------------------------------------------ */
/* Hauptprogramm mit Aufruf des numerischen Lösungsverfahrens.              */
/* ------------------------------------------------------------------------ */
int main(void)
{
    size_t num_steps = 1000;  /* Anzahl Integrationsschritte    */
    size_t dim       = 1;     /* Eine DGL erster Ordnung        */
    double t_begin   = 0;     /* Startzeitpunkt der Simulation  */
    double t_end     = 1.0;   /* Endzeitpunkt der Simulation    */

    dgl_rk4(t_begin, t_end, num_steps, dim, sys, step, init, 0);
    // dgl_euler(t_begin, t_end, num_steps, dim, sys, step, init, 0);
    // dgl_heun(t_begin, t_end, num_steps, dim, sys, step, init, 0);

    return 0;
}
```

2.2. Differentialgleichung zweiter Ordnung

```c
/* ------------------------------------------------------------------------ */
/* Ein Pendel (1.0 m Länge) wird um 45° ausgelenkt und bei t = 0 sec        */
/* losgelassen. Es gilt die folgende Differentialgleichung 2. Ordnung:      */
/*                                                                          */
/* d²phi/dt² = -g/L * sin(phi)       mit g = 9.81 m/s² und L = 1.0 m        */
/*                                                                          */
/* Es wird die Pendelschwingung phi(t) im Intervall 0...10 sec ermittelt,   */
/* die Ausgabe der berechneten Werte erfolgt auf dem Bildschirm.            */
/* ------------------------------------------------------------------------ */
#define _USE_MATH_DEFINES
#include <stdio.h>
#include <math.h>
#include "hmdgl.h"

/* ------------------------------------------------------------------------ */
/* Die oben angegebene Differentialgleichung zweiter Ordnung wird in ein    */
/* System erster Ordnung umgewandelt und in der Funktion <sys> definiert:   */
/*                                                                          */
/* du0/dt = u1                                                              */
/* du1/dt = -g/L * sin(u0)                                                  */
/* ------------------------------------------------------------------------ */
void sys(double dudt[], double t, const double u[], void *pdata)
{
    dudt[0] = u[1];
    dudt[1] = -9.81 / 1.0 * sin(u[0]);
}

/* ------------------------------------------------------------------------ */
/* Nach jedem Integrationsschritt wird die Funktion <step> aufgerufen.      */
/* ------------------------------------------------------------------------ */
void step(double t, const double u[], void *pdata)
{
    /* Aktuelle "Simulationszeit" und Pendelauslenkung ausgeben */
    printf("t = %6.2f sec, phi = %6.1f Grad \n", t, u[0] * 180.0 / M_PI);
}

/* ------------------------------------------------------------------------ */
/* Anfangsbedingung: phi(0 sec) = 45°                                       */
/* ------------------------------------------------------------------------ */
void init(double u[], void *pdata)
{
    u[0] = 45.0 * M_PI / 180.0;
    u[1] = 0;  /* Zunächst ist das Pendel noch in Ruhe */
}

/* ------------------------------------------------------------------------ */
/* Hauptprogramm mit Aufruf des numerischen Lösungsverfahrens.              */
/* ------------------------------------------------------------------------ */
int main(void)
{
    size_t num_steps = 1000;  /* Anzahl der Integrationsschritte  */
    double t_begin   = 0;     /* Startzeitpunkt der Simulation    */
    double t_end     = 10.0;  /* Endzeitpunkt der Simulation      */
    size_t dim       = 2;     /* Zwei Gleichungen erster Ordnung  */

    dgl_rk4(t_begin, t_end, num_steps, dim, sys, step, init, 0);
    // dgl_euler(t_begin, t_end, num_steps, dim, sys, step, init, 0);
    // dgl_heun(t_begin, t_end, num_steps, dim, sys, step, init, 0);

    return 0;
}
```

2.3. Weiterleiten von Daten mittels pdata-Zeiger

```c
/* ------------------------------------------------------------------------ */
/* Wie im vorherigen Beispiel wird eine Pendelschwingung simuliert.         */
/* Die Anfangsauslenkung wird nun allerdings vom Anwender eingegeben        */
/* und mittels "pdata-Zeiger" an die Funktion <init> übergeben.             */
/* ------------------------------------------------------------------------ */
#define _USE_MATH_DEFINES
#include <stdio.h>
#include <math.h>
#include "hmdgl.h"

/* ------------------------------------------------------------------------ */
/* Differentialgleichungssystem aus zwei Gleichungen erster Ordnung.        */
/* ------------------------------------------------------------------------ */
void sys(double dudt[], double t, const double u[], void *pdata)
{
    dudt[0] = u[1];
    dudt[1] = -9.81 / 1.0 * sin(u[0]);
}

/* ------------------------------------------------------------------------ */
/* Nach jedem Integrationsschritt wird die Funktion <step> aufgerufen.      */
/* ------------------------------------------------------------------------ */
void step(double t, const double u[], void *pdata)
{
    /* Aktuelle "Simulationszeit" und Pendelauslenkung ausgeben */
    printf("t = %6.2f sec, phi = %6.1f Grad \n", t, u[0] * 180.0 / M_PI);
}

/* ------------------------------------------------------------------------ */
/* Die Anfangsauslenkung wird mittels "pdata-Zeiger" an <init> übergeben.   */
/* ------------------------------------------------------------------------ */
void init(double u[], void *pdata)
{
    double *p_auslenk = (double*)pdata;
    u[0] = *p_auslenk * M_PI / 180.0;
    u[1] = 0;  /* Zunächst ist das Pendel noch in Ruhe */
}

/* ------------------------------------------------------------------------ */
/* Hauptprogramm mit Aufruf des numerischen Lösungsverfahrens.              */
/* ------------------------------------------------------------------------ */
int main(void)
{
    size_t num_steps = 1000;  /* Anzahl der Integrationsschritte  */
    double t_begin   = 0;     /* Startzeitpunkt der Simulation    */
    double t_end     = 10.0;  /* Endzeitpunkt der Simulation      */
    size_t dim       = 2;     /* Zwei Gleichungen erster Ordnung  */

    double auslenk;
    printf("Anfangsauslenkung in Grad: ");
    scanf("%lf", &auslenk);

    dgl_rk4(t_begin, t_end, num_steps, dim, sys, step, init, &auslenk);
    // dgl_euler(t_begin, t_end, num_steps, dim, sys, step, init, &auslenk);
    // dgl_heun(t_begin, t_end, num_steps, dim, sys, step, init, &auslenk);

    return 0;
}
```

2.4. Programmierung in C++, Variante (a)

HMDGL kann auch in C++-Projekten genauso eingesetzt werden, wie in den Abschnitten 2.1. bis 2.3. gezeigt, also durch Einbinden von hmdgl.c und hmdgl.h und durch Nutzung der darin definierten Funktionen. Alternativ steht in der Datei **hmdgl.hpp** auch eine Template-basierte Variante von HMDGL zur Verfügung, die eine einfachere Programmierung in C++ ermöglicht. Bei dieser Variante sind die Dateien hmdgl.c und hmdgl.h nicht erforderlich.

```
// ---------------------------------------------------------------------------
// Simulation einer Pendelschwingung in C++, Variante (a)
// ---------------------------------------------------------------------------
#define _USE_MATH_DEFINES
#include <iostream>
#include <vector>
#include <math.h>
#include "hmdgl.hpp"

// ---------------------------------------------------------------------------
// Die Zustandsvariablen des DGL-Systems werden in Containern des Typs
// <state_type> gespeichert. Als <state_type> sind grundsätzlich alle
// Container geeignet, die die Methoden begin() und end() besitzen.
// ---------------------------------------------------------------------------
typedef std::vector<double> state_type;
const size_t dim = 2;  // Zwei Gleichungen 1. Ordnung

// ---------------------------------------------------------------------------
// Definition des Differentialgleichungssystems
// ---------------------------------------------------------------------------
state_type sys(double /* t */, state_type u)
{
    state_type dudt(dim);
    dudt[0] = u[1];
    dudt[1] = -9.81 / 1.0 * sin(u[0]);
    return dudt;
}

// ---------------------------------------------------------------------------
// Nach jedem Integrationsschritt: Simulationszeit, Pendelauslenkung ausgeben
// ---------------------------------------------------------------------------
void step(double t, state_type u)
{
    std::cout << t << "\t" << 180.0 * u[0] / M_PI << std::endl;
}

// ---------------------------------------------------------------------------
// Hauptprogramm mit Aufruf des numerischen Lösungsverfahrens
// ---------------------------------------------------------------------------
int main()
{
    size_t num_steps = 1000;  // Anzahl der Integrationsschritte
    double t_begin   = 0;     // Startzeitpunkt der Simulation
    double t_end     = 10.0;  // Endzeitpunkt der Simulation

    state_type init(dim);     // Anfangsauslenkung = 45°
    init[0] = 45.0 * M_PI / 180.0;
    init[1] = 0;

    HMDGL::dgl_rk4(t_begin, t_end, num_steps, sys, step, init);
    // HMDGL::dgl_heun(t_begin, t_end, num_steps, sys, step, init);
    // HMDGL::dgl_euler(t_begin, t_end, num_steps, sys, step, init);
}
```

2.5. Programmierung in C++, Variante (b)

```
// ----------------------------------------------------------------------------
// Simulation einer Pendelschwingung in C++, Variante (b)
//
// In C++-Programmen können die beiden Funktionen <sys> und <step> auch
// direkt im Hauptprogramm definiert werden (sog. Lambda-Funktionen).
// ----------------------------------------------------------------------------
#define _USE_MATH_DEFINES
#include <iostream>
#include <vector>
#include <math.h>
#include "hmdgl.hpp"

// ----------------------------------------------------------------------------
// Die Zustandsvariablen des DGL-Systems werden in Containern des Typs
// <state_type> gespeichert. Als <state_type> sind grundsätzlich alle
// Container geeignet, die die Methoden begin() und end() besitzen.
// ----------------------------------------------------------------------------
typedef std::vector<double> state_type;
const size_t dim = 2;  // Zwei Gleichungen 1. Ordnung

// ----------------------------------------------------------------------------
// Hauptprogramm mit Aufruf des numerischen Lösungsverfahrens.
// ----------------------------------------------------------------------------
int main()
{
    size_t num_steps = 1000;  // Anzahl der Integrationsschritte
    double t_begin   = 0;     // Startzeitpunkt der Simulation
    double t_end     = 10.0;  // Endzeitpunkt der Simulation

    state_type init(dim);     // Anfangsauslenkung = 45°
    init[0] = 45.0 * M_PI / 180.0;
    init[1] = 0;

    // Differentialgleichung zweiter Ordnung (Pendelschwingung)
    auto sys = [] (double /* t */, state_type u)
    {
        state_type dudt(dim);
        dudt[0] = u[1];
        dudt[1] = -9.81 / 1.0 * sin(u[0]);
        return dudt;
    };

    // Nach jedem Integrationsschritt wird <step> aufgerufen:
    // Aktuelle "Simulationszeit" und Pendelauslenkung ausgeben
    auto step = [] (double t, state_type u)
    {
        std::cout << t << "\t" << 180.0 * u[0] / M_PI << std::endl;
    };

    HMDGL::dgl_rk4(t_begin, t_end, num_steps, sys, step, init);
    // HMDGL::dgl_heun(t_begin, t_end, num_steps, sys, step, init);
    // HMDGL::dgl_euler(t_begin, t_end, num_steps, sys, step, init);
}
```

3. Differentialgleichungen höherer Ordnung

Zur Verarbeitung einer Differentialgleichung höherer Ordnung muss diese zunächst in ein System von Differentialgleichungen erster Ordnung umgewandelt werden (siehe Abschnitt 2.2). Das folgende Beispiel zeigt die Umwandung einer Differentialgleichung dritter Ordnung in ein System von drei Differentialgleichungen erster Ordnung:

$$\dddot{y} = \sin(2\pi \cdot t) - 2\ddot{y} + \dot{y} + 2y \qquad y(0) = 1,\ \dot{y}(0) = 0,\ \ddot{y}(0) = 0$$

Dazu werden die neuen Variablen u_0, u_1 und u_2 eingeführt:

$$u_0 = y$$
$$u_1 = \dot{y}$$
$$u_2 = \ddot{y}$$

Nun kann die ursprüngliche Differentialgleichung dritter Ordnung als System von drei Differentialgleichungen erster Ordnung geschrieben werden:

$$\dot{u}_0 = u_1 \qquad u_0(0) = 1$$
$$\dot{u}_1 = u_2 \qquad u_1(0) = 0$$
$$\dot{u}_2 = \sin(2\pi \cdot t) - 2u_2 + u_1 + 2u_0 \qquad u_2(0) = 0$$

Dieses Gleichungssystem lässt sich unmittelbar in C-Quelltext übertragen:

```
// Differentialgleichungssystem
void sys(double dudt[], double t, const double u[], void *pdata)
{
    dudt[0] = u[1];
    dudt[1] = u[2];
    dudt[2] = sin(2 * M_PI * t) - 2 * u[2] + u[1] + 2 * u[0];
}

// Anfangswerte
void init(double u[], void *pdata)
{
    u[0] = 1;
    u[1] = 0;
    u[2] = 0;
}
```

4. Funktionsübersicht

4.1. Programmierung in C

- ```
  typedef void (*dgl_sys_fnc)(
      double dudt[], double t, const double u[], void *pdata);
  ```

  Definition des Differentialgleichungssystems: Eine Callback-Funktion des Typs <dgl_sys_fnc> berechnet die rechte Seite von du/dt = f(t, u) und speichert das Ergebnis im Vektor dudt[]. Im Parameter t wird die aktuelle Simulationszeit übergeben, im Parameter pdata können beliebige – vom Anwender definierte – Daten weitergeleitet werden.

- ```
  typedef void (*dgl_step_fnc)(double t, const double u[], void *pdata);
  ```

  Nach jedem einzelnen Integrationsschritt wird eine Callback-Funktion des Typs <dgl_step_fnc> aufgerufen und die aktuellen Werte von t und u[] übergeben. Im Parameter pdata können beliebige – vom Anwender definierte – Daten weitergeleitet werden.

- ```
  typedef void (*dgl_init_fnc)(double u[], void *pdata);
  ```

  Vor Beginn des eigentlichen Lösungsverfahrens werden die Startwerte der verwendeten Variablen durch eine Callback-Funktion des Typs <dgl_init_fnc> gesetzt. Im Parameter pdata können beliebige – vom Anwender definierte – Daten weitergeleitet werden.

- ```
  void dgl_euler(double t_begin, double t_end, size_t num_steps, size_t dim,
      dgl_sys_fnc sys, dgl_step_fnc step, dgl_init_fnc init, void *pdata);
  ```

  Das Differentialgleichungssystem <sys> wird mit dem einfachen Euler-Verfahren numerisch integriert. Die Startwerte werden durch die Callback-Funktion <init> gesetzt, nach jedem Integrationsschritt wird die Callback-Funktion <step> aufgerufen. Im Parameter pdata können beliebige – vom Anwender definierte – Daten weitergeleitet werden (falls nicht erforderlich: pdata = 0 setzen).

- ```
  void dgl_rk4(double t_begin, double t_end, size_t num_steps, size_t dim,
      dgl_sys_fnc sys, dgl_step_fnc step, dgl_init_fnc init, void *pdata);
  ```

  Das Differentialgleichungssystem <sys> wird mit dem klassischen 4-stufigen Runge-Kutta-Verfahren numerisch integriert. Die Startwerte werden durch die Callback-Funktion <init> gesetzt, nach jedem Integrationsschritt wird die Callback-Funktion <step> aufgerufen. Im Parameter pdata können beliebige – vom Anwender definierte – Daten weitergeleitet werden (falls nicht erforderlich: pdata = 0 setzen).

- ```
  void dgl_heun(double t_begin, double t_end, size_t num_steps, size_t dim,
      dgl_sys_fnc sys, dgl_step_fnc step, dgl_init_fnc init, void *pdata);
  ```

  Das Differentialgleichungssystem <sys> wird mit dem Heun-Verfahren numerisch integriert. Die Startwerte werden durch die Callback-Funktion <init> gesetzt, nach jedem einzelnen Integrationsschritt wird die Callback-Funktion <step> aufgerufen. Über den Parameter pdata können beliebige – vom Anwender definierte – Daten weitergeleitet werden (falls nicht erforderlich: pdata = 0 setzen).

4.2. Programmierung in C++

Vorbemerkung: Die Zustandsvariablen des Differentialgleichungssystems werden in Containern des Typs <state_type> gespeichert. Als <state_type> sind grundsätzlich alle Container geeignet, die die Methoden begin() und end() besitzen, zum Beispiel std::vector<double>.

- ```
  template<class sys_fn, class step_fn, class state_type>
  void dgl_euler(double t_begin, double t_end, size_t num_steps,
      sys_fn sys, step_fn step, state_type init);
  ```

  Das Differentialgleichungssystem <sys> wird mit dem einfachen Euler-Verfahren numerisch gelöst. Die Anfangswerte werden in <init> übergeben, nach jedem Integrationsschritt wird die Funktion <step> aufgerufen.
  ```

- ```
  template<class sys_fn, class step_fn, class state_type>
  void dgl_heun(double t_begin, double t_end, size_t num_steps,
      sys_fn sys, step_fn step, state_type init);
  ```

  Das Differentialgleichungssystem <sys> wird mit dem Heun-Verfahren numerisch gelöst. Die Anfangswerte werden in <init> übergeben, nach jedem Integrationsschritt wird die Funktion <step> aufgerufen.

- ```
  template<class sys_fn, class step_fn, class state_type>
  void dgl_rk4(double t_begin, double t_end, size_t num_steps,
      sys_fn sys, step_fn step, state_type init);
  ```

  Das Differentialgleichungssystem <sys> wird mit dem klassischen vierstufigen Runge-Kutta-Verfahren numerisch gelöst. Die Anfangswerte werden in <init> übergeben, nach jedem Integrationsschritt wird die Funktion <step> aufgerufen.

5. Kontakt, Lizenz

Tilman Küpper
tilman.kuepper@hm.edu

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

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