Ausarbeitung Übung 3

Höttinger, Kucher

FH Technikum Wien, Game Engineering und Simulation, Wien, AUT

Bsp2

Approximieren der Lösung für ein Anfangswertproblem

Bei einem Anfangswertproblem hat man einen Anfangswert und kann die Ableitung der Funktion mit dem echten Funktionswert berechnen.

$$y(0) = y_0 \tag{1}$$
$$y(t) = f(t) \tag{2}$$
$$\dot{y}(t) = f(y(t)) \tag{3}$$

a) Lösen Sie das System aus Aufgabe 1.) mit dem expliziten Euler-Verfahren für den Fall $\delta = 1$. Integrieren Sie von $t = 0$ bis $t = 1$ mit vorgegebener Schrittweite h. Anmerkung: Für Ihren Euler-Code ist h ein vorzugebender Parameter, $h = 1/n$, wenn n Schritte von $t = 0$ bis $t = 1$ durchgeführt werden.
b) Experimentieren Sie mit $h = 1/2$) (2 Schritte), $h = 1/4$ (4 Schritte), $h = 1/8$ (8 Schritte), usw. Aus Aufgabe 1.) kennen Sie die exakte Lösung an der Stelle $t = 1$ und können daher vergleichen. Wie verhält sich der Approximationsfehler in Abhängigkeit von h?

Das explizite Euler Verfahren ist ein recht einfaches Verfahren um den Funktionswert $y(t)$ zu approximieren.

Die Grundidee besteht darin dass die erste Ableitung mit dem Anfangswert berechnet werden kann und dann damit einen weiterer Funktionswert approximiert werden kann, mit dem wieder die erste Ableitung berechnet werden kann. Die Genauigkeit ist hier abhängig von der Anzahl der Zwischenwerte. Damit bestimmen wir einfach für wie lange der Wert der ersten Ableitung gültig ist um den Funktionswert zu berechnen. Je kleiner wir die Schritte wählen desto genauer wird das Ergebnis.

$$\dot{y} = f(t, y) \tag{4}$$
$$y(t_0) = y_0 \tag{5}$$
$$y_{k+1} = y_k + h * f(t_k, y_k) \tag{6}$$
$$t_k = t_0 + k * h \tag{7}$$

wobei h die Schrittweite ist und k geht von 0 bis zu dem gewünschten t_k.

Die Genauigkeit dieses Verfahren kann nun anhand eines Beispiels demonstriert werden:

$$y(y) = (y_1(t), y_2(t)) \tag{8}$$
$$\dot{y}(t) = A * y(t) \tag{9}$$
$$A = \begin{pmatrix} -1 & 0 \\ 1 & -1 \end{pmatrix} \tag{10}$$
$$y(0) = (1, 1) \tag{11}$$
$$h = 2^{-n} \tag{12}$$

und $y(1)$ soll bestimmt werden. Die exakte Lösung ist:

$y(1) = (0.3678794412, 0.7357588823)$

Schrittweite	y1	y2	Fehler	Fehlerreduktion
0.500000	0.250000	0.750000	0.051819	0.000000
0.250000	0.316406	0.738281	0.024475	0.472324
0.125000	0.343609	0.736305	0.011862	0.484662
0.062500	0.356074	0.735887	0.005839	0.492217
0.031250	0.362055	0.735790	0.002897	0.496097
0.015625	0.364987	0.735766	0.001443	0.498047
0.007813	0.366438	0.735761	0.000720	0.499023
0.003906	0.367160	0.735759	0.000360	0.499512
0.001953	0.367520	0.735759	0.000180	0.499756
0.000977	0.367700	0.735759	0.000090	0.499878
0.000488	0.367790	0.735759	0.000045	0.499939
0.000244	0.367835	0.735759	0.000022	0.499969
0.000122	0.367857	0.735759	0.000011	0.499985
0.000061	0.367868	0.735759	0.000006	0.499992
0.000031	0.367874	0.735759	0.000003	0.499996
0.000015	0.367877	0.735759	0.000001	0.499998

Tabelle 1: Euler Verfahren

In Tabelle 1 ist erkennbar, dass sich der Fehler ungefähr halbiert wenn wir die Schrittanzahl verdoppeln (Schrittweite verringern).

c) Analog zu b); verwenden Sie jedoch das klassische Runge-Kutta Verfahren. Vergleichen Sie die Approximationsqualität mit der des Euler-Verfahrens.

Das Runge-Kutta Verfahren ist ein weiterer Ansatz um ein Anfangswertproblem zu approximieren. Das klassische Runge-Kutta Verfahren ist eine Spezialform des Runge-Kutta Verfahrens, nämlich ein 4-Stufiges explizites Runge-Kutta Verfahren. Dabei wird Euler Teilschritt verwendet um Hillfswerte zu generieren. Mit diesen kann dann wieder der nächste Hilfswert bestimmt werden. Bei dem klassischen Runge-Kutta Verfahren werden 4 Hilfswerte generiert.

$$\dot{y} = f(t, y) \tag{13}$$
$$y(t_0) = y_0 \tag{14}$$
$$k_1 = f(t_k, y_k) \tag{15}$$
$$k_2 = f(t_k + \frac{h}{2}, y_k + \frac{h}{2} * k_1) \tag{16}$$
$$k_3 = f(t_k + \frac{h}{2}, y_k + \frac{h}{2} * k_2) \tag{17}$$
$$k_4 = f(t_k + h, y_k + h * k_3) \tag{18}$$
$$\tag{19}$$

Mit diesen Hilfswerten wird dann der Funktionswert bestimmt:

$$y_{k+1} = y_k + h * \frac{1}{6} * (k_1 + 2k_2 + 2k_3 + k_4) \tag{20}$$
$$\tag{21}$$

Schrittweite	y1	y2	Fehler	Fehlerreduktion
0.500000	0.368171	0.734762	0.000353	0.000000
0.250000	0.367894	0.735712	0.000016	0.046195
0.125000	0.367880	0.735756	0.000001	0.053612
0.062500	0.367879	0.735759	0.000000	0.057846
0.031250	0.367879	0.735759	0.000000	0.060117
0.015625	0.367879	0.735759	0.000000	0.061294
0.007813	0.367879	0.735759	0.000000	0.061893
0.003906	0.367879	0.735759	0.000000	0.062205
0.001953	0.367879	0.735759	0.000000	0.062447
0.000977	0.367879	0.735759	0.000000	0.078067
0.000488	0.367879	0.735759	-0.000000	-0.047619
0.000244	0.367879	0.735759	-0.000000	1.333333
0.000122	0.367879	0.735759	0.000000	-2.250000
0.000061	0.367879	0.735759	-0.000000	-0.555556
0.000031	0.367879	0.735759	0.000000	-1.200000
0.000015	0.367879	0.735759	-0.000000	-30.500000

Tabelle 2: Runge-Kutta Verfahren

Hier sehen wir, dass das klassische Runge-Kutta Verfahren bei der gleichen Schrittweite wesentlich bessere Ergebnisse liefert. Zudem verringert sich der Fehler auf ein $\frac{1}{16}$ wenn wir die Schrittanzahl verdoppeln.

Matlab Code:

Listing 1: Approximation mit Euler-Verfahren

```
phi = 1;              %
t_max = 1;            % Integrationsweite
y_start = [1; 1];     % Anfangsbedingung y1(t=0), y2(t=0)
result = zeros(15, 9);
% Exakt
y1_exact = exp(-t_max);
y2_exact = exp(-t_max) + phi * t_max * exp(-t_max);

% Explizites Euler Verfahren
for n=1:16            % Anzahl der Schritte
    h = 2^(-n);       % Schrittweite  1, 1/2, 1/4, 1/8, ...

    y1_approx = y_start(1);
    y2_approx = y_start(2);

    for i=h:h:t_max
        y1_abl = -y1_approx;
        y2_abl = phi * y1_approx - y2_approx;

        % Funktionswert = Alter Wert + h * Ableitung
        y1_approx = y1_approx + h * y1_abl;
        y2_approx = y2_approx + h * y2_abl;
    end
    % Durchschnittlicher Fehler
    error = y1_exact - y1_approx;
    error = error + y2_exact - y2_approx;
    error = error / 2;

    result(n, 1) = h;
    result(n, 2) = y1_approx;
    result(n, 3) = y2_approx;
    result(n, 4) = error;

    % Fehlerreduktion
    if n > 1
        last_error =  result(n - 1, 4);
        result(n, 5) = error / last_error;
    end
end
```

Listing 2: Approximation mit Runge-Kutta-Verfahren

```
% Klassisches Runge-Kutta Verfahren
for n=1:16          % Anzahl der Schritte
    h = 2^(-n);     % Schrittweite  1, 1/2, 1/4, 1/8, ...

    y1_approx = y_start(1);
    y2_approx = y_start(2);

    for i=h:h:t_max
        % y1
        k1 = -y1_approx;
        k2 = -(y1_approx + h/2 * k1);
        k3 = -(y1_approx + h/2 * k2);
        k4 = -(y1_approx + h * k3);

        y1_approx_new = y1_approx + h * (1/6) * (k1 + (2*k2↩
            ) + (2*k3) + k4);

        % y2
        k1 = phi * y1_approx - y2_approx;
        k2 = phi * -k2 - (y2_approx + h/2 * k1);
        k3 = phi * -k3 - (y2_approx + h/2 * k2);
        k4 = phi * -k4 - (y2_approx + h * k3);

        y2_approx = y2_approx + h * (1/6) * (k1 + (2*k2) + ↩
            (2*k3) + k4);
        y1_approx = y1_approx_new;
    end
    % Durchschnittlicher Fehler
    error = y1_exact - y1_approx;
    error = error + y2_exact - y2_approx;
    error = error / 2;

    result(n, 6) = y1_approx;
    result(n, 7) = y2_approx;
    result(n, 8) = error;

    % Fehlerreduktion
    if n > 1
        last_error =  result(n - 1, 8);
        result(n, 9) = error / last_error;
    end
end
```