Numerik

Ingenieurinformatik Teil 2, Sommersemester 2026

David Straub

Gliederung

1. Einführung in Matlab
2. Arbeiten mit Arrays
3. Funktionen und Kontrollstrukturen
4. Analysis
5. Lineare Algebra
6. **Differentialgleichungen**
7. Einführung in Simulink

Fahrplan: Differentialgleichungen

Einheit 1 → Motivation und Herleitung am Ingenieurbeispiel → Begriffe: Ordnung, Anfangswertproblem → Analytische vs. numerische Lösung → Euler'sches Polygonzugverfahren in Matlab

Einheit 2 – Heute → Standardform für numerische Verfahren → Numerische Lösungsverfahren → Lösung in Matlab

Einheit 3 → Anwendungen

Standardform

Rückblick: Was kann das Euler-Verfahren?

Aus Einheit 1: Das Euler-Verfahren löst DGLn der Form

$$\dot{y} = f(t,\, y), \qquad y(t_0) = y_0$$

Schrittweise Approximation mit Schrittweite h:

$$y_{i+1} = y_i + h \cdot f(t_i,\, y_i)$$

Konkret – die Batterie-DGL: $f(t, T) = \dfrac{P - \lambda\,(T - T_\infty)}{C_\text{th}}$

Frage: Was passiert, wenn wir **mehrere gekoppelte** DGLn haben – oder eine DGL **höherer Ordnung**?

Erweiterung 1: Gekoppelte DGLn 1. Ordnung

Zwei Zellen in einem Modul, thermisch gekoppelt (μ = Kopplung zwischen den Zellen):

$$C_\text{th}\,\dot{T}_1 = P_1 - \lambda(T_1 - T_\infty) - \mu(T_1 - T_2)$$
$$C_\text{th}\,\dot{T}_2 = P_2 - \lambda(T_2 - T_\infty) - \mu(T_2 - T_1)$$

Beide DGLn sind 1. Ordnung, aber **gekoppelt** – $\dot{T}_1$ hängt von T_2 ab und umgekehrt.

Als Vektor geschrieben hat das genau die gleiche Form wie vorher:

$$\dot{y} = f(t,\, y), \qquad y = \begin{pmatrix} T_1 \\ T_2 \end{pmatrix}$$

Das Euler-Verfahren funktioniert **unverändert** – nur mit Vektoren statt Skalaren.

Die Standardform

Das Euler-Verfahren (und alle anderen Löser) arbeiten mit der **Standardform**:

$$\boxed{\dot{y} = f(t,\, y), \qquad y(t_0) = y_0}$$

- $y(t) \in \mathbb{R}^n$: Zustandsvektor
- f: vektorwertige Funktion der Zeit und des Zustands
- Skalarer Fall ($n = 1$): die bekannte Form aus Einheit 1

Offene Frage: Was ist mit DGLn *höherer Ordnung* – z. B. Schwingungsgleichungen?

Erweiterung 2: DGL 2. Ordnung → Standardform

Eine DGL 2. Ordnung enthält $\ddot{x}$ – das passt nicht direkt in die Standardform.

Beispiel – Schwingungsgleichung: $m\ddot{x} + kx = 0$

Idee: Neue Variable $v := \dot{x}$ einführen, dann gilt $\dot{v} = \ddot{x}$.

Die eine DGL 2. Ordnung wird zu **zwei** DGLn 1. Ordnung:

$$\dot{x} = v, \qquad \dot{v} = -\frac{k}{m}\, x$$

In Vektorschreibweise – jetzt wieder Standardform!

$$\dot{y} = \underbrace{\begin{pmatrix} v \\ -\dfrac{k}{m}\, x \end{pmatrix}}_{=f(t,\,y)}, \qquad y = \begin{pmatrix} x \\ v \end{pmatrix}$$

Standardform: Vorgehensweise

Wie viele Komponenten hat y? → Gleich der **Ordnung** der DGL.

Vorgehen:

1. DGL nach der **höchsten Ableitung** auflösen: $\ddot{x} = ...$
2. Neue Variablen einführen: $y_1 := x, \quad y_2 := \dot{x}$
3. System aufschreiben: $\dot{y}_1 = y_2, \quad \dot{y}_2 = ...$

4. Anfangsbedingungen als Vektor: $y_0 = \begin{pmatrix} x(0) \\ \dot{x}(0) \end{pmatrix}$

? Aufgabe: Standardform

Schreiben Sie die folgende DGL in ein System von DGLn 1. Ordnung um. Geben Sie auch die Anfangsbedingungen als Vektor y_0 an.

Gedämpfte Schwingung mit äußerer Anregung:

$$m\ddot{x} + d\,\dot{x} + k\,x = F_0 \cos(\omega t), \qquad x(0) = 0, \quad \dot{x}(0) = v_0$$

Wie viele Komponenten hat y?

Numerische Lösungsverfahren

Euler-Verfahren: Einfluss der Schrittweite

Wir haben das Euler-Verfahren bereits kennengelernt – aber wie gut funktioniert es wirklich?

Testfall: Federschwinger ohne Dämpfung

$$\ddot{x} + x = 0 \quad \Rightarrow \quad \dot{y} = \begin{pmatrix} v \\ -x \end{pmatrix}, \quad y_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Exakte Lösung: $x(t) = \cos(t)$ – eine gleichmäßige Schwingung, **keine wachsende Amplitude**.

Frage: Was passiert, wenn wir die Schrittweite h variieren?

Euler-Verfahren: Grenzen – Live-Demo

```
f = @(t, y) [y(2); -y(1)];   % Federschwinger: y = [x; v]
dt = 0.001;
t = 0:dt:30;
y = zeros(2, length(t));
y(:,1) = [1; 0];              % x(0)=1, v(0)=0
for i = 1:length(t)-1
    y(:,i+1) = y(:,i) + dt * f(t(i), y(:,i));
end
plot(t, y(1,:), t, cos(t), '--')
legend('Euler', 'Exakt'),  grid on
```

Euler-Verfahren: Fazit

- Kleine Schrittweite → genaue Lösung, aber **langsam**
- Große Schrittweite → Lösung wächst oder **explodiert**
- Beim Federschwinger wird Energie künstlich erzeugt – physikalisch falsch

→ Wir brauchen ein Verfahren, das **genauer** ist bei gleicher Schrittweite.

Von Euler zu Runge-Kutta

Euler nutzt nur die Steigung am **Anfang** des Intervalls:

$$k_1 = f(t_i, y_i), \qquad y_{i+1} = y_i + dt \cdot k_1$$

→ Analogie: **Rechteckregel** aus der Numerischen Integration.

Modifiziertes Euler-Verfahren: Erst einen halben Schritt, Steigung dort auswerten, dann ganzen Schritt:

$$k_1 = f(t_i, y_i), \quad k_2 = f\big(t_i + \tfrac{dt}{2},\ y_i + \tfrac{dt}{2}\, k_1\big)$$

$$y_{i+1} = y_i + dt \cdot k_2$$

→ Analogie: **Mittelpunktregel**. Genauer als Euler, weil die Steigung in der Mitte repräsentativer ist.

RK3 und die Simpsonregel

RK3: Steigung am Anfang, in der Mitte und am Ende:

$$k_1 = f(t_i, y_i), \quad k_2 = f\big(t_i + \tfrac{dt}{2},\ y_i + \tfrac{dt}{2} k_1\big), \quad k_3 = f(t_i + dt,\ y_i + dt\, k_2)$$

$$y_{i+1} = y_i + \frac{dt}{6}(k_1 + 4k_2 + k_3)$$

→ Gewichte $\frac{1}{6}, \frac{4}{6}, \frac{1}{6}$ – das ist die **Simpsonregel** (Analysis II / Keplersche Fassregel), angewendet auf die Steigungsfunktion f.

RK4: Noch besser

RK4: Die Mitte wird **zweimal** ausgewertet – k_3 verbessert die Schätzung von k_2:

$$k_1 = f(t_i,\ y_i)$$

$$k_2 = f\left(t_i + \tfrac{dt}{2},\ y_i + \tfrac{dt}{2}\,k_1\right)$$

$$k_3 = f\left(t_i + \tfrac{dt}{2},\ y_i + \tfrac{dt}{2}\,k_2\right)$$

$$k_4 = f\left(t_i + dt,\ y_i + dt\,k_3\right)$$

$$y_{i+1} = y_i + \frac{dt}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)$$

Euler als wiederverwendbare Funktion

Unser Euler-Code:

```
f  = @(t, y) [y(2); -y(1)];
dt = 0.01;
t  = 0:dt:20;
y  = zeros(2, length(t));
y(:,1) = [1; 0];
for i = 1:length(t)-1
    y(:,i+1) = y(:,i) + dt * f(t(i), y(:,i));
end
```

Dieser Code braucht nur drei Dinge: die **DGL** `f`, den **Zeitbereich** und die **Anfangsbedingung**.

→ Das Verfahren selbst lässt sich als Funktion kapseln:

Euler als Funktion

```
function [t, y] = ode_euler(f, tspan, y0, dt)
    t = tspan(1):dt:tspan(2);
    y = zeros(length(y0), length(t));  % funktioniert für 1D und nD
    y(:,1) = y0;
    for i = 1:length(t)-1
        y(:,i+1) = y(:,i) + dt * f(t(i), y(:,i));
    end
end
```

- `tspan` ist der Zeitbereich `[t0, tend]`
- `f` ist ein Function-Handle

ode45 in MATLAB

MATLAB löst DGLn in Standardform mit `ode45` – der Name steht für **Runge-Kutta der Ordnung 4/5**: - Zwei Verfahren (RK4 und RK5) laufen parallel – die Differenz schätzt den Fehler - `ode45` passt dt automatisch an

```
[t, y] = ode45(f, tspan, y0)
```

Argument	Bedeutung	Beispiel
`f`	Function-Handle für $f(t, y)$	`@(t,y) [y(2); -y(1)]`
`tspan`	Zeitbereich `[t0, tend]`	`[0, 20]`
`y0`	Anfangsbedingung (Spaltenvektor)	`[1; 0]`
`t`	Zeitpunkte (Spaltenvektor)	
`y`	Lösungsmatrix: eine Spalte pro Variable	`y(:,1)` $= x$, `y(:,2)` $= v$

Federschwinger mit ode45

```
f = @(t, y) [y(2); -y(1)];   % y = [x; v]

[t, y] = ode45(f, [0, 20], [1; 0]);

plot(t, y(:,1), 'b', t, y(:,2), 'r')
legend('x(t)', 'v(t)')
xlabel('t'),  grid on
```

`y(:,1)` enthält $x(t)$, `y(:,2)` enthält $v(t)$ – eine Zeile pro Zeitpunkt.

Aufgabe: Schwingung mit ode45

Lösen Sie die **gedämpfte Schwingung** aus der vorherigen Aufgabe numerisch:

$$m\ddot{x} + d\,\dot{x} + k\,x = F_0\cos(\omega t), \qquad x(0) = 0, \quad \dot{x}(0) = 1$$

mit $m = 1$, $d = 0{,}2$, $k = 4$, $F_0 = 1$, $\omega = 2$.

1. Schreiben Sie die Funktion `f(t, y)` in MATLAB
2. Lösen Sie mit `ode45` für $t \in [0, 30]$
3. Plotten Sie $x(t)$ und $v(t)$

Was beobachten Sie im Langzeitverhalten?