

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** (Gleichungssysteme, Eigenwerte, ...)
6. **Differentialgleichungen**
7. **Einführung in Simulink**

4. Analysis

Warum Analysis in der Numerik?

Ingenieurprobleme führen fast immer auf Funktionen:

Problem	Mathematik	Matlab
Wo reißt das Bauteil?	Nullstelle einer Kennlinie	<code>roots</code> , <code>fzero</code>
Wie viel Energie wird verbraucht?	Integral einer Leistungskurve	<code>integral</code>
Wie stark ändert sich die Kraft?	Ableitung einer Funktion	<code>polyder</code> , <code>diff</code>
Modell aus Messdaten gewinnen	Kurvenanpassung	<code>polyfit</code>

Ziel dieser Einheit: Funktionen in Matlab beschreiben und damit rechnen.

Fahrplan – 2 Einheiten

Einheit 1 (heute): Wie beschreibe ich eine Funktion in Matlab? - Polynome – strukturierte Darstellung und Rechnen damit - Function Handles & Anonymous Functions – Funktionen als Objekte

Einheit 2: Was rechne ich mit Funktionen? - Numerische Integration - Numerische Differentiation - Nullstellensuche

Polynome

Polynome – das häufigste Ingenieurmodell

Materialkennlinien, Biegelinien, Sensor-Kalibrierungen – viele physikalische Zusammenhänge lassen sich durch ein Polynom annähern:

$$p(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$$

Beispiel: Federkennlinie (nichtlinear)

$$F(x) = 500x - 80x^2$$

Problem: Wie werte ich $F(x)$ effizient für viele Werte aus? Und wie berechne ich Ableitung oder Integral?

Matlab speichert Polynome als Koeffizientenvektor – allerdings **nicht** in der Reihenfolge, die man aus der Mathematik kennt.

Achtung: Reihenfolge ist umgekehrt zur Mathematik

In der Mathematik stehen die Koeffizienten **aufsteigend** (niedrigster Grad zuerst):

$$p(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n \longrightarrow [a_0, a_1, \dots, a_n]$$

Matlab (und NumPy) speichern sie **absteigend** (höchster Grad zuerst):

$$p(x) = a_nx^n + \dots + a_1x + a_0 \longrightarrow [a_n, \dots, a_1, a_0]$$

Beispiel: $p(x) = 1 - 2x + 3x^2$

Konvention	Vektor
Mathematik (aufsteigend)	[1, -2, 3]
Matlab (absteigend)	[3, -2, 1]

Eselsbrücke: In Matlab steht der **höchste Grad zuerst** – wie man ein Polynom beim Aufschreiben liest: $3x^2 - 2x + 1$.

Darstellung als Koeffizientenvektor

Ein Polynom wird in Matlab durch den **absteigenden** Koeffizientenvektor dargestellt:

$$p(x) = 3x^2 - 2x + 1 \longrightarrow [3, -2, 1]$$

$$F(x) = 500x - 80x^2 \longrightarrow [-80, 500, 0]$$

Warum? Weil Ableitung und Integral von Polynomen direkt auf den Koeffizienten operieren – kein symbolisches Rechnen nötig:

$$p'(x) = a_n \cdot n \cdot x^{n-1} + \dots + a_1 \longrightarrow \text{polyder}(p)$$

Die Länge des Vektors bestimmt den Grad: `length(p) - 1`.

polyval – Polynom auswerten

```
p = [-80, 500, 0];           % F(x) = -80x^2 + 500x

x = 0:0.01:5;
F = polyval(p, x);           % wertet p für alle x aus

plot(x, F)
xlabel('Auslenkung x [m]')
ylabel('Kraft F [N]')
title('Federkennlinie')
```

`polyval(p, x)` ist äquivalent zu `p(1)*x.^2 + p(2)*x + p(3)` – aber: - funktioniert für jeden Grad ohne Formelanpassung - kombiniert mit `roots`, `polyder`, `polyint`

roots – Nullstellen berechnen

Problem: Ab welcher Auslenkung ist die Kraft null (Feder entspannt)?

```
p = [-80, 500, 0];
nullstellen = roots(p)
```

```
nullstellen =
    6.2500
         0
```

Matlab löst das **intern numerisch** – es berechnet die Eigenwerte der *Begleitmatrix* des Polynoms. Das verbindet sich später mit dem Kapitel Lineare Algebra.

`roots` liefert alle n Nullstellen (auch komplexe). Physikalisch relevant sind nur reelle Nullstellen im sinnvollen Bereich.

polyder und polyint – Ableitung und Integral

Ableitung = lokale Empfindlichkeit / Steigung der Kennlinie:

```
p = [-80, 500, 0];
dp = polyder(p)      % dp = [-160, 500]
```

$$F'(x) = -160x + 500$$

Integral = Fläche unter der Kurve (z. B. gespeicherte Energie):

```
P = polyint(p)      % P = [-26.6667, 250, 0, 0]
```

$$\int F(x) dx = -\frac{80}{3}x^3 + 250x^2 + C$$

Bestimmtes Integral: `polyval(P, x2) - polyval(P, x1)`

Was passiert hier?

Berechnen Sie **von Hand** die Ableitung des Polynoms:

$$p(x) = 4x^3 - 6x^2 + 2$$

Was liefert `polyder([4, -6, 0, 2])`?

Überprüfen Sie anschließend in Matlab.

Bonus: Was ergibt `roots(polyder([4, -6, 0, 2]))`? Was bedeutet das geometrisch?

Polynome – Ausprobieren

Gegeben: Federkennlinie $F(x) = 500x - 80x^2$

```
p = [-80, 500, 0];
```

1. Plotten Sie $F(x)$ für $x \in [0, 6]$.
2. Bei welchen Auslenkungen ist $F(x) = 0$? ($\rightarrow$ `roots`)
3. Wie groß ist die Steigung $F'(x)$ bei $x = 2$? ($\rightarrow$ `polyder` + `polyval`)
4. Berechnen Sie die gespeicherte Energie $\int_0^3 F(x) dx$. ($\rightarrow$ `polyint` + `polyval`)

Polynom-Funktionen: Matlab vs. NumPy

Operation	Matlab	NumPy
Koeffizientenvektor	<code>p = [-80, 500, 0]</code>	<code>p = np.array([-80, 500, 0])</code>
Auswerten	<code>polyval(p, x)</code>	<code>np.polyval(p, x)</code>
Nullstellen	<code>roots(p)</code>	<code>np.roots(p)</code>
Ableitung	<code>polyder(p)</code>	<code>np.polyder(p)</code>
Stammfunktion	<code>polyint(p)</code>	<code>np.polyint(p)</code>
Kurvenanpassung	<code>polyfit(x, y, n)</code>	<code>np.polyfit(x, y, n)</code>

Die NumPy-Funktionen sind bewusst kompatibel zu Matlab – gleiche Konventionen, gleiche Koeffizientenreihenfolge. Wer das in Matlab versteht, kann es direkt in Python übertragen.

Function Handles und Anonymous Functions

Problem: Plotten wird umständlich

Bisher: Funktion plotten = Vektor aufbauen, auswerten, plotten:

```
x = linspace(0, 2*pi, 500);
y = exp(-x) .* sin(x);
plot(x, y)
```

Das funktioniert – aber: Bereich und Auflösung müssen manuell gewählt werden, und die Funktion ist nicht wiederverwendbar.

fplot löst das Problem – aber fplot erwartet kein Array, sondern ein **Funktionsobjekt**:

```
fplot(@x exp(-x) .* sin(x), [0, 2*pi])
```

`@x` ... erzeugt ein solches Funktionsobjekt – einen **Function Handle**.

Function Handle – Funktion als Objekt

Ein **Function Handle** ist eine Variable, die auf eine Funktion zeigt:

```
f = @sin;           % Handle auf eingebaute Funktion
f(pi/2)             % → 1
```

```
g = @exp;
g(0)                % → 1
```

- `f` ist eine normale Variable – kann gespeichert, übergeben, in Arrays gespeichert werden
- `f(x)` ruft die Funktion auf

In Python ist das dasselbe: `f = math.sin` speichert eine Funktion ohne sie aufzurufen. In Matlab schreibt man `@sin`.

Anonymous Functions

Für eigene Ausdrücke ohne separate `.m`-Datei:

```
f = @(x) x.^2 + 1;
f(3)                % → 10
```

```
f([1, 2, 3])      % → [2, 5, 10]
```

Mehrere Argumente:

```
g = @(x, y) sqrt(x.^2 + y.^2);
g(3, 4)          % → 5
```

	Anonymous Function	.m-Datei
Länge	Einzeiler	beliebig komplex
Geltungsbereich	lokal im Skript	überall aufrufbar
Wann?	kurze Ausdrücke	mehrere Zeilen, Fallunterscheidungen

Punkt-Operator ist Pflicht

fplot und andere Funktionen übergeben **Vektoren** – ohne `.^`, `.*`, `./` rechnet Matlab mit Matrixoperationen und wirft einen Fehler:

```
f = @(x) x^2 + 1;      %  funktioniert nicht für Vektoren
f = @(x) x.^2 + 1;     %  elementweise – immer so schreiben
```

```
g = @(x) sin(x) .* exp(-x); %  beide Operanden vektorisiert
```

Faustregel: Jedes `*`, `^`, `/` in einer Anonymous Function bekommt einen Punkt davor.

Closure – Funktionen mit Parametern

integral, fzero, ode45 erwarten eine Funktion mit **einem** Argument.

Was tun, wenn die Physik mehr Parameter hat?

```
rho = 1.225;
g    = 9.81;
druck = @(h) rho * g * h; % rho und g werden eingefroren

integral(druck, 0, 1000) % druck(h) hat nur ein Argument – passt
```

rho und g werden beim **Erstellen** als Kopie eingefroren.

Spätere Änderungen im Workspace haben keinen Effekt:

```
rho = 0;           % zu spät  
druck(100)        % → 1201.25, nicht 0
```

Was passiert hier? 

```
a = 2;  
f = @() a * x;  
a = 10;  
f(3)
```

Was gibt $f(3)$ aus – 6 oder 30?