Programmieren – Praktikum 6

Ingenieurinformatik Teil 1, Wintersemester 2026/27

David Straub

Diagramme und eigene Datentypen

Heute verbinden Sie Diagramme mit dem simulierten Pack – und bauen die Zelle als eigenen Datentyp. Falls Sie `zelle.py` nicht mehr haben: https://raw.githubusercontent.com/DavidMStraub/teaching-materials/main/programmieren/zelle.py (neben Ihr Skript legen).

Zur Erinnerung: `lese_spannung(nummer, ladezustand, strom_A)` liefert die Spannung von Zelle `nummer` (1 bis 8) bei `ladezustand` von 1.0 (voll) bis 0.0 (leer) und Entladestrom `strom_A`. Die Zellen haben eine Kapazität von `kapazitaet_Ah = 3.5`; ein Zeitschritt ist `zeitschritt_h = 0.1`.

Aufgabe 1: Eine Ladekurve plotten

1. Simulieren Sie die Entladung einer Zelle Ihrer Wahl wie in Praktikum 3: `ladezustand` startet bei `1.0` und sinkt pro Schritt um `strom_A * zeitschritt_h / kapazitaet_Ah`. Sammeln Sie Ladezustand und Spannung diesmal in zwei Listen statt sie nur auszugeben
2. Plotten Sie die Kurve (`plt.plot`) mit Achsenbeschriftung und Gitter
3. Markieren Sie den Punkt, an dem die Zelle unter 3,0 V fällt, mit einem roten Punkt

Aufgabe 2: Zwei Ströme im Vergleich

1. Wiederholen Sie die Simulation für zwei unterschiedliche Ströme derselben Zelle
2. Plotten Sie beide Kurven in einem Diagramm, mit unterschiedlichem Stil (z. B. `"b-"` und `"r--"`)
3. Titel per `if`: Wählen Sie eine sinnvolle Schwelle für den Unterschied der Endspannungen und lassen Sie den Titel entsprechend wählen

Aufgabe 3: Die Zelle als eigener Datentyp

1. Schreiben Sie eine Klasse `Batteriezelle` mit `__init__(self, nummer)`, die die Zellnummer speichert
2. Ergänzen Sie eine Methode `spannung(self, ladezustand, strom_A)`, die `lese_spannung` für diese Zelle aufruft und das Ergebnis zurückgibt (`return`)
3. Erzeugen Sie ein Exemplar und geben Sie die Spannung bei zwei verschiedenen Ladezuständen aus

Aufgabe 4: Verhalten ergänzen

1. Ergänzen Sie eine Methode `bewertung(self, ladezustand, strom_A)`, die ein Urteil zurückgibt (`return`): unter 3,0 V ⟶ `kritisch`, sonst ⟶ `ok`
2. Erzeugen Sie Exemplare für mehrere Zellen (Schleife über Zellnummern) und geben Sie für jede das Urteil bei einem Ladezustand und Strom Ihrer Wahl aus
3. Was hat sich gegenüber der Funktionslösung aus Praktikum 3 verändert – was ist gleich geblieben?

Zusatz für Schnelle

- Ergänzen Sie eine Methode `kapazitaet_bis_grenze(self, strom_A, grenze_V)`, die wie in Praktikum 3 zurückgibt, bei welchem Ladezustand die Grenze unterschritten wird
- Plotten Sie mehrere Zellen in einem Diagramm – geben Sie jeder Linie ein `label="Zelle X"` mit, `plt.legend()` zeigt dann eine Legende