API-Dokumentation des Energiespeicher-Simulationssystems

Till Jonas Wellkamp

10. August 2025

Inhaltsverzeichnis

1 Projektübersicht

Dieses Dokument enthält die vollständige API-Dokumentation des Energiespeicher-Simulationssystems im PEP 8 konformen Docstring-Format. Das System simuliert den optimierten Betrieb eines Batteriespeichers auf Basis historischer Strompreisdaten.

1.1 Modulstruktur

- `models/` - Kernmodelle (Energiespeicher)
- `strategies/` - Handelsstrategien
- `data/` - Datenlader und -verarbeitung
- `visualization/` - Visualisierungsmodule
- `economic_analysis/` - Wirtschaftlichkeitsanalyse
- `utils/` - Hilfsfunktionen

2 Hauptmodul: main.py

```
"""
Hauptmodul f r die Energiespeicher-Simulation.

Dieses Modul koordiniert die gesamte Simulation eines Batteriespeichers
zur Arbitrage im Strommarkt. Es l dt historische Preisdaten, f hrt die
Optimierung durch und generiert umfassende Analysen und Visualisierungen.

Attributes
----------
OUTPUT_DIR : str
    Basisverzeichnis f r alle Ausgabedateien
DEFAULT_PRICE_FILE : str
    Standardpfad zur CSV-Datei mit Preisdaten

Examples
--------
Ausf hrung  ber  Kommandozeile:

    $ python main.py --file prices.csv --output results/

Oder direkt in Python:

    >>> from main import main
    >>> output_dir = main()
"""

def main() -> str:
    """
    F hrt die komplette Energiespeicher-Simulation aus.

    Diese Funktion orchestriert den gesamten Simulationsprozess:
    1. Initialisierung und Konfiguration
    2. Laden der Strompreisdaten
    3. Ausf hrung der Handelsstrategie
    4. Wirtschaftlichkeitsanalyse
    5. Erstellung von Visualisierungen
    6. Speichern aller Ergebnisse

    Returns
    -------
    str
        Pfad zum Ausgabeverzeichnis mit allen Simulationsergebnissen

    Raises
    ------
    FileNotFoundError
```

```
        Wenn die Preisdatei nicht gefunden wird
    ValueError
        Bei ung ltigen Konfigurationsparametern

    Notes
    -----
    Die Simulation verwendet eine perzentil-basierte Handelsstrategie
    mit Rolling-Window-Ansatz. Alle Parameter sind in config.py
    konfigurierbar.

    Die Laufzeit betr gt typischerweise 30-60 Sekunden f r ein
    Jahr Preisdaten mit 15-Minuten-Aufl sung.

    See Also
    --------
    config : Konfigurationsmodul mit allen Parametern
    threshold_lookahead : Implementierung der Handelsstrategie
    run_economic_analysis : Wirtschaftlichkeitsberechnung
    """
    # Implementierung...


def save_config_to_output(output_dir: str) -> None:
    """
    Speichert die aktuelle Konfiguration f r Reproduzierbarkeit.

    Parameters
    ----------
    output_dir : str
        Zielverzeichnis f r die Konfigurationsdatei

    Notes
    -----
    Erstellt eine Datei 'config_snapshot.py' mit allen verwendeten
    Parametern zum Zeitpunkt der Simulation.
    """
    # Implementierung...
```

3 Kernmodell: models/energy_storage.py

```
"""
Modul zur Modellierung eines Energiespeichers.

Dieses Modul implementiert die physikalischen und wirtschaftlichen
Eigenschaften eines Batteriespeichers, einschlie lich Lade-/Entladelogik,
Effizienzverlusten und Transaktionsverfolgung.
"""

class EnergyStorage:
    """
    Modelliert einen Batteriespeicher f r Energiehandel.

    Diese Klasse repr sentiert einen Energiespeicher mit definierten
    technischen Parametern und verwaltet dessen Zustand w hrend der
    Simulation. Sie unterst tzt kontinuierliche Lade- und Entladevorg nge
     ber  mehrere Zeitintervalle.

    Parameters
    ----------
    capacity_mwh : float
        Speicherkapazit t in Megawattstunden (MWh)
    charge_rate : float
        C-Rate des Speichers (Verh ltnis Leistung zu Kapazit t)
    efficiency : float
        Wirkungsgrad des Speichers (0 < efficiency <= 1)
    fee_per_mwh : float
        Transaktionsgeb hr in Euro pro MWh

    Attributes
```

```
    ----------
    energy_level : float
        Aktueller Ladezustand in MWh
    cash : float
        Kumulierter Kassenbestand in Euro
    transactions : list of dict
        Historie aller durchgef hrten Transaktionen
    is_charging : bool
        Flag f r aktiven Ladevorgang
    is_discharging : bool
        Flag f r aktiven Entladevorgang
    total_cycles : float
        Anzahl der   quivalenten   Vollzyklen

    Examples
    --------
    >>> storage = EnergyStorage(
    ...     capacity_mwh=10.0,
    ...     charge_rate=0.5,
    ...     efficiency=0.9,
    ...     fee_per_mwh=2.0
    ... )
    >>> storage.start_charging(price=50.0, time_index=0)
    >>> storage.continue_process(time_index=1)

    Notes
    -----
    Die Klasse berechnet automatisch Degradationskosten basierend
    auf der Anzahl der Zyklen (konfigurierbar in config.py).
    """

    def __init__(self, capacity_mwh: float, charge_rate: float,
                 efficiency: float, fee_per_mwh: float):
        """Initialisiert den Energiespeicher mit gegebenen Parametern."""
        # Implementierung...

    def start_charging(self, price: float, time_index: int) -> None:
        """
        Startet einen neuen Ladevorgang.

        Parameters
        ----------
        price : float
            Aktueller Strompreis in Euro/MWh
        time_index : int
            Zeitindex des Vorgangsbeginns

        Raises
        ------
        RuntimeError
            Wenn bereits ein Vorgang l uft
        """
        # Implementierung...

    def start_discharging(self, price: float, time_index: int) -> None:
        """
        Startet einen neuen Entladevorgang.

        Parameters
        ----------
        price : float
            Aktueller Strompreis in Euro/MWh
        time_index : int
            Zeitindex des Vorgangsbeginns

        Raises
        ------
        RuntimeError
            Wenn bereits ein Vorgang l uft
        """
        # Implementierung...

```

```
    def continue_process(self, time_index: int) -> None:
        """
        Setzt den aktuellen Lade-/Entladevorgang fort.

        Parameters
        ----------
        time_index : int
            Aktueller Zeitindex

        Notes
        -----
        Diese Methode wird automatisch in jedem Zeitschritt
        aufgerufen, wenn ein Vorgang aktiv ist.
        """
        # Implementierung...

    def is_processing(self) -> bool:
        """
        Pr ft, ob ein Lade- oder Entladevorgang l uft.

        Returns
        -------
        bool
            True wenn Vorgang aktiv, sonst False
        """
        return self.is_charging or self.is_discharging

    def finalize(self) -> None:
        """
        Schlie t die Simulation ab und berechnet finale Metriken.

        Diese Methode sollte am Ende der Simulation aufgerufen werden,
        um Degradationskosten zu berechnen und die finale Bilanz zu
        erstellen.

        Notes
        -----
        Berechnet NPV (Net Present Value) unter Ber cksichtigung von:
        - Kumulierten Handelsgewinnen
        - Degradationskosten
        - Restenergie im Speicher
        """
        # Implementierung...
```

4 Handelsstrategie: strategies/threshold_lookahead.py

```
"""
Modul f r perzentil-basierte Handelsstrategie mit Vorausschau.

Implementiert eine adaptive Schwellenwertstrategie, die auf der
statistischen Analyse zuk nftiger Preise basiert.
"""

def threshold_lookahead(
    prices_df: pd.DataFrame,
    storage: EnergyStorage,
    window_size: int = None
) -> tuple[EnergyStorage, list]:
    """
    F hrt perzentil-basierte Handelsstrategie mit Vorausschau aus.

    Diese Funktion implementiert eine dynamische Handelsstrategie,
    die f r jeden Zeitpunkt ein Vorausschau-Fenster analysiert und
    Kauf-/Verkaufsentscheidungen basierend auf Preis-Perzentilen trifft.

    Parameters
    ----------
    prices_df : pd.DataFrame
        DataFrame mit Spalten 'price' und 'datetime'
```

```
    storage : EnergyStorage
        Instanz des Energiespeichers
    window_size : int, optional
        Gr   e des Vorausschau-Fensters in Zeitintervallen.
        Standard: ROLLING_WINDOW_SIZE aus config.py

    Returns
    -------
    tuple[EnergyStorage, list]
        - Aktualisiertes storage-Objekt mit allen Transaktionen
        - Liste mit Energiehistorie f r jeden Zeitpunkt

    Algorithm
    ---------
    F r jeden Zeitpunkt t:
    1. Analysiere Preise im Fenster [t, t+window_size]
    2. Berechne 20. und 80. Perzentil als Schwellenwerte
    3. Kaufentscheidung wenn:
        - Aktueller Preis < 20. Perzentil
        - Zuk nftiger Durchschnitt > aktueller Preis
        - Speicher hat Kapazit t
    4. Verkaufsentscheidung wenn:
        - Aktueller Preis > 80. Perzentil
        - Zuk nftiger Durchschnitt < aktueller Preis
        - Speicher hat Energie

    Notes
    -----
    Die Strategie ber cksichtigt zus tzlich:
    - Vermeidung von Folgeverlusten durch Preisvergleiche
    - Dynamische Volumenanpassung basierend auf Preisextremit t
    - Mindesthandelsvolumen zur Vermeidung von Kleinstransaktionen
    - Kontinuierliche Vorg nge  ber  mehrere Zeitschritte

    Examples
    --------
    >>> import pandas as pd
    >>> from models.energy_storage import EnergyStorage
    >>>
    >>> # Preisdaten vorbereiten
    >>> prices_df = pd.read_csv('prices.csv')
    >>>
    >>> # Speicher initialisieren
    >>> storage = EnergyStorage(
    ...     capacity_mwh=10.0,
    ...     charge_rate=0.5,
    ...     efficiency=0.9,
    ...     fee_per_mwh=2.0
    ... )
    >>>
    >>> # Strategie ausf hren
    >>> final_storage, history = threshold_lookahead(
    ...     prices_df,
    ...     storage,
    ...     window_size=96  # 24 Stunden bei 15-Min-Intervallen
    ... )
    >>>
    >>> print(f"Finaler Gewinn: {final_storage.cash:.2f} Euro")

    See Also
    --------
    EnergyStorage : Speichermodell
    config.ROLLING_WINDOW_SIZE : Standard-Fenstergr  e
    """
    # Implementierung...
```

5 Datenverarbeitung: data/data_loader.py

```
"""
```

```
Modul zum Laden und Verarbeiten von Strompreisdaten.

Bietet Funktionen zum Einlesen, Validieren und Vorverarbeiten
von CSV-Dateien mit historischen Strompreisen.
"""

def load_price_data(filepath: str) -> pd.DataFrame:
    """
    L dt Strompreisdaten aus einer CSV-Datei.

    Parameters
    ----------
    filepath : str
        Pfad zur CSV-Datei mit Preisdaten

    Returns
    -------
    pd.DataFrame
        DataFrame mit verarbeiteten Preisdaten
        Enth lt Spalten: 'datetime', 'date', 'price'

    Raises
    ------
    FileNotFoundError
        Wenn die Datei nicht existiert
    ValueError
        Bei ung ltigem Dateiformat

    Notes
    -----
    Erwartet CSV-Format mit Spalten:
    - 'Start (CET/CEST)': Zeitstempel
    - 'Day-Ahead Auction (DE-LU)': Preis in Euro/MWh

    Die Funktion f hrt folgende Verarbeitungen durch:
    - Parsing von Datums-/Zeitangaben
    - Umgang mit fehlenden Werten
    - Sortierung nach Zeit
    - Hinzuf gen von Datumsspalte f r Gruppierung

    Examples
    --------
    >>> df = load_price_data('data/smard_2023.csv')
    >>> print(f"Geladene Daten: {len(df)} Zeitpunkte")
    >>> print(f"Zeitraum: {df['datetime'].min()} bis {df['datetime'].max()}")
    """
    # Implementierung...
```

6 Wirtschaftsanalyse: economic_analysis/analyzer.py

```
"""
Modul f r wirtschaftliche Analyse der Speichersimulation.

Berechnet Kennzahlen wie NPV, IRR, Amortisationszeit und
erstellt detaillierte Wirtschaftlichkeitsberichte.
"""

def run_economic_analysis(
    storage: EnergyStorage,
    energy_history: list,
    prices_df: pd.DataFrame,
    output_dir: str = None
) -> dict:
    """
    F hrt umfassende Wirtschaftlichkeitsanalyse durch.

    Parameters
    ----------
    storage : EnergyStorage
```

```
        Finaler Speicherzustand nach Simulation
    energy_history : list
        Historie der Energielevel  ber  Zeit
    prices_df : pd.DataFrame
        Urspr ngliche Preisdaten
    output_dir : str, optional
        Verzeichnis f  r  Ausgabedateien

    Returns
    -------
    dict
        Wirtschaftliche Kennzahlen:
        - 'npv': Net Present Value in Euro
        - 'irr': Internal Rate of Return in Prozent
        - 'payback_period': Amortisationszeit in Jahren
        - 'total_revenue': Gesamterl se in Euro
        - 'total_costs': Gesamtkosten in Euro
        - 'profit_margin': Gewinnmarge in Prozent
        - 'daily_results': Liste t glicher Ergebnisse

    Notes
    -----
    Die Analyse ber cksichtigt:
    - Investitionskosten (CAPEX)
    - Betriebskosten (OPEX)
    - Degradationskosten
    - Transaktionsgeb hren
    - Zeitwert des Geldes (Diskontierung)

    Generiert automatisch:
    - Excel-Report mit allen Kennzahlen
    - Visualisierungen der Ergebnisse
    - T gliche Performance-Metriken

    Examples
    --------
    >>> analysis = run_economic_analysis(
    ...     storage=final_storage,
    ...     energy_history=history,
    ...     prices_df=prices,
    ...     output_dir='results/'
    ... )
    >>> print(f"NPV: {analysis['npv']:,.2f} Euro")
    >>> print(f"IRR: {analysis['irr']:.1f}%")

    See Also
    --------
    calculate_npv : NPV-Berechnung
    calculate_irr : IRR-Berechnung
    create_excel_report : Excel-Report-Generierung
    """
    # Implementierung...

def calculate_npv(
    cash_flows: list[float],
    discount_rate: float = 0.05
) -> float:
    """
    Berechnet den Net Present Value (Kapitalwert).

    Parameters
    ----------
    cash_flows : list[float]
        Liste der Cashflows pro Periode
    discount_rate : float, optional
        Diskontierungssatz (Standard: 5%)

    Returns
    -------
    float
        Net Present Value in Euro

```

```
    Formula
    -------
    """
```

$$\text{NPV} = \sum_{t=0}^{n} \frac{\text{CF}_t}{(1+r)^t}$$

wobei:

- CF_t = Cashflow in Periode t
- r = Diskontierungssatz
- n = Anzahl Perioden

```
    """
    # Implementierung...
```

7 Visualisierung: visualization/plotting.py

```
"""
Modul f r Visualisierungen der Simulationsergebnisse.

Erstellt verschiedene Diagramme zur Analyse der Speicherperformance,
Handelsstrategien und wirtschaftlichen Ergebnisse.
"""

def visualize_day(
    result: dict,
    output_path: str = None,
    show_plot: bool = False
) -> None:
    """
    Erstellt Tagesvisualisierung der Speicheraktivit t.

    Parameters
    ----------
    result : dict
        Tagesergebnis mit Keys:
        - 'date': Datum
        - 'prices': Preisdaten
        - 'energy_levels': Speicherst nde
        - 'transactions': Transaktionen
    output_path : str, optional
        Pfad zum Speichern der Grafik
    show_plot : bool, optional
        Ob Plot angezeigt werden soll (Standard: False)

    Notes
    -----
    Erstellt mehrteiliges Diagramm mit:
    - Strompreisverlauf
    - Speicherstand
    - Lade-/Entladevorg nge
    - Gewinn/Verlust pro Transaktion
    """
    # Implementierung...

def plot_cycle_analysis(
    daily_results: list[dict],
    output_path: str = None
) -> None:
    """
    Analysiert und visualisiert Batteriezyklen.

    Parameters
    ----------
    daily_results : list[dict]
```

```
        Liste t glicher Simulationsergebnisse
    output_path : str, optional
        Basispfad f r Ausgabedateien

    Creates
    -------
    Mehrere Visualisierungen:
    - Zyklen  ber  Zeit
    - Zyklenverteilung
    - Degradationsanalyse
    - Lebensdauerprognose

    Notes
    -----
    Ein Vollzyklus entspricht einmaligem vollst ndigen
    Laden und Entladen der Batteriekapazit t.
    """
    # Implementierung...

def plot_charge_discharge_patterns(
    daily_results: list[dict],
    output_path: str = None
) -> None:
    """
    Visualisiert Lade- und Entlademuster.

    Parameters
    ----------
    daily_results : list[dict]
        Liste t glicher Simulationsergebnisse
    output_path : str, optional
        Basispfad f r Ausgabedateien

    Creates
    -------
    Heatmap-Visualisierungen:
    - St ndliche Aktivit tsmuster
    - Wochentags-Analyse
    - Saisonale Muster
    - Preiskorrelationen
    """
    # Implementierung...

def plot_efficiency_analysis(
    daily_results: list[dict],
    output_path: str = None
) -> None:
    """
    Analysiert Speichereffizienz und -auslastung.

    Parameters
    ----------
    daily_results : list[dict]
        Liste t glicher Simulationsergebnisse
    output_path : str, optional
        Basispfad f r Ausgabedateien

    Metrics
    -------
    - Kapazit tsauslastung in Prozent
    - Effizienzverluste in MWh
    - Durchschnittliche Zykleneffizienz
    - State-of-Charge Verteilung
    """
    # Implementierung...

def plot_price_arbitrage_analysis(
    daily_results: list[dict],
    output_path: str = None
) -> None:
    """
    Analysiert Preisarbitrage-Performance.
```

```
121
122     Parameters
123     ----------
124     daily_results : list[dict]
125         Liste t glicher Simulationsergebnisse
126     output_path : str, optional
127         Basispfad f r Ausgabedateien
128
129     Analysis
130     --------
131     - Durchschnittliche Kauf-/Verkaufspreise
132     - Handelsmarge (Spread)
133     - Preisvolatilit t vs. Gewinn
134     - Optimale Handelszeiten
135     """
136     # Implementierung...
```

8 Grid Search: grid-search_simulation.py

```
1  """
2  Modul f r Grid-Search-Optimierung der Handelsparameter.
3
4  F hrt systematische Parametersuche durch, um optimale
5  Konfiguration f r maximalen Gewinn zu finden.
6  """
7
8  def run_continuous_simulation_for_window(
9      prices_df: pd.DataFrame,
10     window_size: int,
11     output_dir: str,
12     save_plots: bool = True,
13     save_daily_plots: bool = False
14 ) -> dict:
15     """
16     F hrt Simulation f r eine spezifische Window-Gr   e aus.
17
18     Parameters
19     ----------
20     prices_df : pd.DataFrame
21         Preisdaten
22     window_size : int
23          Gr   e des Vorausschau-Fensters
24     output_dir : str
25         Ausgabeverzeichnis
26     save_plots : bool, optional
27         Ob Plots gespeichert werden sollen
28     save_daily_plots : bool, optional
29         Ob t gliche Plots erstellt werden sollen
30
31     Returns
32     -------
33     dict
34         Simulationsergebnisse:
35         - 'window_size': Verwendete Fenstergr  e
36         - 'profit': Gesamtgewinn
37         - 'npv': Net Present Value
38         - 'cycles': Anzahl Zyklen
39         - 'transactions': Anzahl Transaktionen
40
41     Notes
42     -----
43     Diese Funktion wird typischerweise in einer Schleife
44     f r verschiedene window_size Werte aufgerufen.
45     """
46     # Implementierung...
47
48 def test_window_sizes_with_full_visualization(
49     window_sizes: list[int] = None
50 ) -> pd.DataFrame:
```

```
    """
    Testet verschiedene Window-Gr   en mit Visualisierung.

    Parameters
    ----------
    window_sizes : list[int], optional
        Liste zu testender Fenstergr  en
        Standard: [1, 2, 4, 8, 16, 32, 64, 96, 192]

    Returns
    -------
    pd.DataFrame
        Vergleichstabelle aller getesteten Konfigurationen

    Creates
    -------
    - Vergleichsdiagramme
    - Optimierungskurven
    - Parameter-Sensitivit tsanalyse
    - Empfehlung f  r optimale Konfiguration

    Examples
    --------
    >>> results_df = test_window_sizes_with_full_visualization()
    >>> optimal_window = results_df.loc[results_df['profit'].idxmax(), 'window_size']
    >>> print(f"Optimale Window-Gr   e: {optimal_window}")
    """
    # Implementierung...
```

9 Hilfsfunktionen: utils/

9.1 utils/logging_setup.py

```
"""
Modul f  r Logging-Konfiguration.

Richtet strukturiertes Logging f  r die gesamte Anwendung ein.
"""

def setup_logging(level: str = 'INFO') -> logging.Logger:
    """
    Konfiguriert das Logging-System.

    Parameters
    ----------
    level : str, optional
        Logging-Level ('DEBUG', 'INFO', 'WARNING', 'ERROR')
        Standard: 'INFO'

    Returns
    -------
    logging.Logger
        Konfigurierter Root-Logger

    Notes
    -----
    Erstellt sowohl Konsolen- als auch Datei-Handler mit
    unterschiedlichen Formatierungen f  r optimale Lesbarkeit.
    """
    # Implementierung...
```

9.2 utils/localization.py

```
"""
Modul f  r deutsche Lokalisierung.

Bietet Funktionen zur Formatierung von Datums- und Zeitangaben
```

```
in deutscher Sprache.
"""

def setup_german_locale() -> bool:
    """
    Versucht deutsche Lokalisierung zu aktivieren.

    Returns
    -------
    bool
        True wenn erfolgreich, sonst False

    Notes
    -----
    Falls Systemlokalisierung fehlschl gt, werden interne
      bersetzungstabellen    verwendet.
    """
    # Implementierung...

def format_date_german(
    date: datetime,
    format_str: str = "%d.%m.%Y"
) -> str:
    """
    Formatiert Datum in deutschem Format.

    Parameters
    ----------
    date : datetime
        Zu formatierendes Datum
    format_str : str, optional
        Format-String (Standard: TT.MM.JJJJ)

    Returns
    -------
    str
        Formatiertes Datum

    Examples
    --------
    >>> from datetime import datetime
    >>> date = datetime(2024, 3, 15)
    >>> print(format_date_german(date))
    '15.03.2024'
    """
    # Implementierung...

def get_german_weekday(date: datetime) -> str:
    """
    Gibt deutschen Wochentag zur ck.

    Parameters
    ----------
    date : datetime
        Datum

    Returns
    -------
    str
        Deutscher Wochentag

    Examples
    --------
    >>> from datetime import datetime
    >>> date = datetime(2024, 3, 15)  # Freitag
    >>> print(get_german_weekday(date))
    'Freitag'
    """
    # Implementierung...

def get_german_month(date: datetime) -> str:
    """
```

```
    Gibt deutschen Monatsnamen zur ck.

    Parameters
    ----------
    date : datetime
        Datum

    Returns
    -------
    str
        Deutscher Monatsname

    Examples
    --------
    >>> from datetime import datetime
    >>> date = datetime(2024, 3, 15)
    >>> print(get_german_month(date))
    'M rz'
    """
    # Implementierung...
```

10 Konfiguration: config.py

```
"""
Zentrale Konfigurationsdatei f r die Energiespeicher-Simulation.

Alle wichtigen Parameter und Einstellungen sind hier zentral definiert
und k nnen f r verschiedene Simulationsszenarien angepasst werden.
"""

# Batteriespeicher-Parameter
# -------------------------
CAPACITY_MWH = 10.0
"""float: Speicherkapazit t in Megawattstunden (MWh)."""

CHARGE_RATE = 0.5
"""float: C-Rate des Speichers (0.5 = 2 Stunden f r Vollladung)."""

EFFICIENCY = 0.9
"""float: Round-Trip-Effizienz des Speichers (90%)."""

FEE_PER_MWH = 2.0
"""float: Transaktionsgeb hr in Euro pro MWh."""

# Wirtschaftliche Parameter
# ------------------------
CAPEX_PER_MWH = 150000
"""float: Investitionskosten in Euro pro MWh Kapazit t."""

OPEX_PER_YEAR = 10000
"""float: J hrliche Betriebskosten in Euro."""

DISCOUNT_RATE = 0.05
"""float: Diskontierungssatz f r NPV-Berechnung (5%)."""

BATTERY_LIFETIME_YEARS = 15
"""int: Erwartete Batterielebensdauer in Jahren."""

MAX_CYCLES = 5000
"""int: Maximale Anzahl Vollzyklen  ber  Lebensdauer."""

# Handelsstrategie-Parameter
# -------------------------
ROLLING_WINDOW_SIZE = 96
"""int: Standard-Fenstergr  e f r Vorausschau (96 * 15min = 24h)."""

BUY_PERCENTILE = 20
"""int: Perzentil f r Kaufschwelle (20. Perzentil)."""

```

```
SELL_PERCENTILE = 80
"""int: Perzentil f r Verkaufsschwelle (80. Perzentil)."""

MIN_TRADE_VOLUME = 0.5
"""float: Mindesthandelsvolumen in MWh."""

MIN_SOC = 0.1
"""float: Minimaler State-of-Charge (10%)."""

MAX_SOC = 0.9
"""float: Maximaler State-of-Charge (90%)."""

# Daten und Ausgabe
# -----------------
DEFAULT_PRICE_FILE = 'data/smard_2023_prices.csv'
"""str: Standardpfad zur Preisdatei."""

OUTPUT_DIR = 'output'
"""str: Basisverzeichnis f r Ausgabedateien."""

FIGURE_DPI = 300
"""int: Aufl sung f r gespeicherte Grafiken (DPI)."""

FIGURE_FORMAT = 'png'
"""str: Dateiformat f r Grafiken ('png', 'pdf', 'svg')."""

# Logging
# -------
LOG_LEVEL = 'INFO'
"""str: Standard-Logging-Level ('DEBUG', 'INFO', 'WARNING', 'ERROR')."""

LOG_FORMAT = '%(asctime)s - %(name)s - %(levelname)s - %(message)s'
"""str: Format f r Log-Nachrichten."""

# Simulation
# ----------
SIMULATION_TIMESTEP_MINUTES = 15
"""int: Zeitschrittweite in Minuten (15-Minuten-Intervalle)."""

PARALLEL_PROCESSING = True
"""bool: Ob Parallelverarbeitung verwendet werden soll."""

MAX_WORKERS = 4
"""int: Maximale Anzahl paralleler Worker-Prozesse."""

# Degradation
# -----------
def calculate_degradation_cost(cycles: float) -> float:
    """
    Berechnet Degradationskosten basierend auf Zyklenanzahl.

    Parameters
    ----------
    cycles : float
        Anzahl der Vollzyklen

    Returns
    -------
    float
        Degradationskosten in Euro

    Notes
    -----
    Verwendet lineares Degradationsmodell:
    Kosten = (cycles / MAX_CYCLES) * CAPEX * CAPACITY_MWH

    Examples
    --------
    >>> cost = calculate_degradation_cost(100)
    >>> print(f"Degradationskosten f r 100 Zyklen: {cost:.2f} Euro")
    """
    total_capex = CAPEX_PER_MWH * CAPACITY_MWH
```

```
    degradation_rate = cycles / MAX_CYCLES
    return degradation_rate * total_capex
```

11 Verwendungsbeispiele

11.1 Basis-Simulation

```
"""
Beispiel f r eine einfache Simulation mit Standardparametern.
"""

import pandas as pd
from models.energy_storage import EnergyStorage
from strategies.threshold_lookahead import threshold_lookahead
from data.data_loader import load_price_data
from economic_analysis.analyzer import run_economic_analysis
import config

def run_basic_simulation():
    """
    F hrt eine Basis-Simulation durch.

    Returns
    -------
    dict
        Simulationsergebnisse mit Gewinn und Kennzahlen

    Examples
    --------
    >>> results = run_basic_simulation()
    >>> print(f"Gesamtgewinn: {results['profit']:.2f} Euro")
    >>> print(f"NPV: {results['npv']:.2f} Euro")
    >>> print(f"Anzahl Zyklen: {results['cycles']:.1f}")
    """
    # 1. Preisdaten laden
    prices_df = load_price_data(config.DEFAULT_PRICE_FILE)

    # 2. Speicher initialisieren
    storage = EnergyStorage(
        capacity_mwh=config.CAPACITY_MWH,
        charge_rate=config.CHARGE_RATE,
        efficiency=config.EFFICIENCY,
        fee_per_mwh=config.FEE_PER_MWH
    )

    # 3. Handelsstrategie ausf hren
    final_storage, energy_history = threshold_lookahead(
        prices_df=prices_df,
        storage=storage,
        window_size=config.ROLLING_WINDOW_SIZE
    )

    # 4. Wirtschaftsanalyse
    analysis = run_economic_analysis(
        storage=final_storage,
        energy_history=energy_history,
        prices_df=prices_df
    )

    return {
        'profit': final_storage.cash,
        'npv': analysis['npv'],
        'cycles': final_storage.total_cycles,
        'transactions': len(final_storage.transactions)
    }
```

11.2 Parameter-Optimierung

```python
"""
Beispiel f r Parameteroptimierung mit Grid Search.
"""

def optimize_window_size():
    """
    Optimiert die Window-Gr   e f r maximalen Gewinn.

    Returns
    -------
    tuple[int, float]
        Optimale Window-Gr   e und zugeh riger Gewinn

    Examples
    --------
    >>> optimal_window, max_profit = optimize_window_size()
    >>> print(f"Optimale Window-Gr   e: {optimal_window}")
    >>> print(f"Maximaler Gewinn: {max_profit:.2f} Euro")
    """
    window_sizes = [1, 2, 4, 8, 16, 32, 64, 96, 192]
    results = []

    for window_size in window_sizes:
        # Simulation f r jede Window-Gr   e
        profit = simulate_with_window(window_size)
        results.append((window_size, profit))
        print(f"Window {window_size}: {profit:.2f} Euro")

    # Beste Konfiguration finden
    optimal = max(results, key=lambda x: x[1])
    return optimal
```

12 API-Referenz Zusammenfassung

12.1 Hauptklassen

- `EnergyStorage`: Kernklasse für Batteriespeicher-Modellierung
- `EconomicAnalyzer`: Wirtschaftlichkeitsberechnungen
- `Visualizer`: Erstellung von Diagrammen und Reports

12.2 Hauptfunktionen

- `main()`: Haupteinstiegspunkt der Simulation
- `threshold_lookahead()`: Implementierung der Handelsstrategie
- `run_economic_analysis()`: Wirtschaftlichkeitsanalyse
- `load_price_data()`: Laden der Preisdaten
- `visualize_day()`: Tagesvisualisierung

12.3 Konfigurationsparameter

- Batteriespeicher: Kapazität, C-Rate, Effizienz
- Wirtschaftlich: CAPEX, OPEX, Diskontrate
- Strategie: Window-Größe, Perzentile, Mindestvolumen
- Simulation: Zeitschritte, Parallelverarbeitung

13 Anhang: Type Hints

```python
"""
 bersicht   der verwendeten Type Hints nach PEP 484.
"""

from typing import Dict, List, Tuple, Optional, Union, Any
import pandas as pd
import numpy as np
from datetime import datetime

# Basis-Typen
Price = float  # Preis in Euro/MWh
Energy = float  # Energie in MWh
Money = float  # Geldbetrag in Euro
TimeIndex = int  # Zeitindex (0-basiert)

# Komplexe Typen
TransactionDict = Dict[str, Union[str, float, int]]
EnergyHistoryEntry = Dict[str, Union[int, float]]
DailyResult = Dict[str, Any]

# Funktions-Signaturen
SimulationResult = Tuple['EnergyStorage', List[EnergyHistoryEntry]]
AnalysisResult = Dict[str, Union[float, List[DailyResult]]]

# DataFrame-Typen
PriceDataFrame = pd.DataFrame  # Mit Spalten: 'datetime', 'price'
ResultsDataFrame = pd.DataFrame  # Mit Spalten variabel

# Array-Typen
PriceArray = np.ndarray[np.float64]
EnergyArray = np.ndarray[np.float64]
```