

# Generierung des Eingangssingals für Barrier Bucket RF Systeme and der GSI



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

**Jonas Christ, Artem Moskalew, Maximilian Nolte  
Jens Harzheim, M.Sc.**

Projektseminar Beschleunigertechnik



Institut für Theorie  
Elektromagnetischer Felder  
Computational Electromagnetics  
Research Group at GSCE

# Outline

---

- 1 Einführung
  - Aufbau
- 2 Der Code
  - Die Blöcke
  - Vorgehensweise
  - Test Driven Development
  - Mock-System
- 3 Optimierung
  - Optimierung der Übertragungsfunktion
- 4 Ausblick

---

# Problemstellung

---

- Barrier-Bucket System

# Problemstellung

---

- Barrier-Bucket System :
  - Longitudinale Manipulation der Bunches

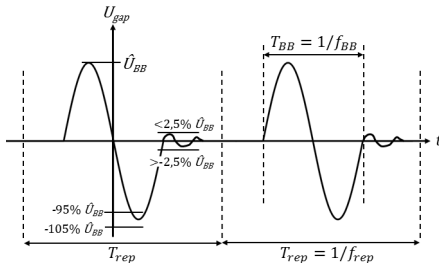
# Problemstellung

---

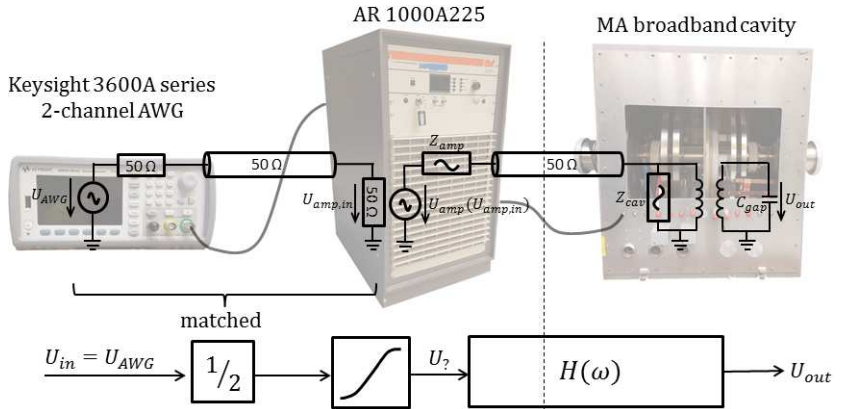
- Barrier-Bucket System :
  - Longitudinale Manipulation der Bunches
- Ziel

# Problemstellung

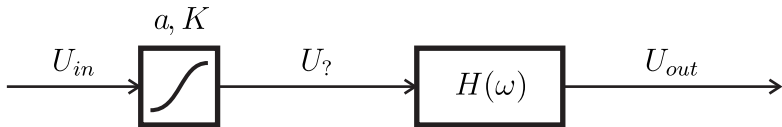
- Barrier-Bucket System :
  - Longitudinale Manipulation der Bunches
- Ziel :
  - Gap Spannung in Form einer Ein-Sinus Periode
  - Qualität des Signals



# Aufbau und Modell



# Aufbau und Modell

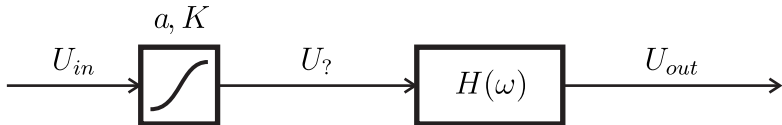




# Aufbau und Modell

■ Gegeben:

- Lineare Übertragungsfunktion  $H$  bestimmt durch Pseudorauschen
- System linear bis  $\hat{U}_{BB} \approx 550 \text{ V}$  genähert



# Aufbau und Modell

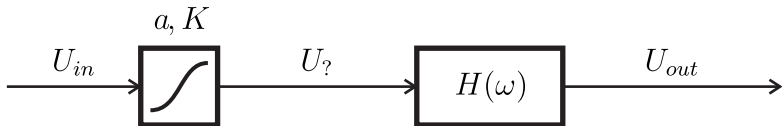
- Gegeben:

- Lineare Übertragungsfunktion  $H$  bestimmt durch Pseudorauschen
- System linear bis  $\hat{U}_{BB} \approx 550 \text{ V}$  genähert

- Hammerstein Modell :

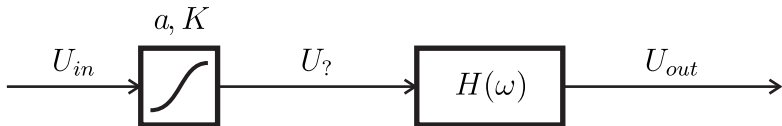
- Ergänzung um eine nichtlineare Vorverzerrung mit einem Potenzreihenansatz

$$U_?(t) = \sum_{n=1}^N a_n [U_{in}(t)]^n \quad \underline{U}_{out}(\omega) = H(\omega) \cdot \underline{U}_?(\omega)$$

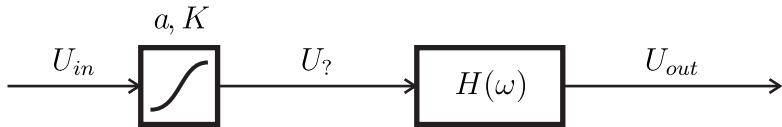


# Aufbau und Modell

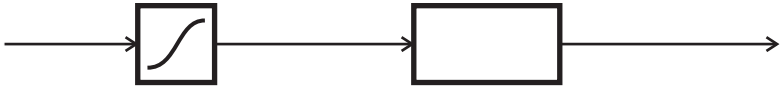
- Gegeben:
  - Lineare Übertragungsfunktion  $H$  bestimmt durch Pseudorauschen
  - System linear bis  $\hat{U}_{BB} \approx 550 \text{ V}$  genähert
- Hammerstein Modell :
  - Ergänzung um eine nichtlineare Vorverzerrung mit einem Potenzreihenansatz
$$U_?(t) = \sum_{n=1}^N a_n [U_{in}(t)]^n \quad \underline{U}_{out}(\omega) = H(\omega) \cdot \underline{U}_?(\omega)$$
- Zielsetzung :
  - Parameter  $a_n$  der Kennlinie  $K$  zu bestimmen



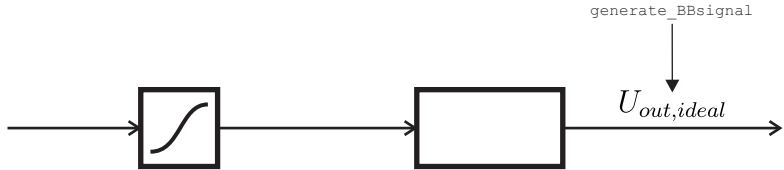
## Code: Die Bausteine



## Code: Die Bausteine

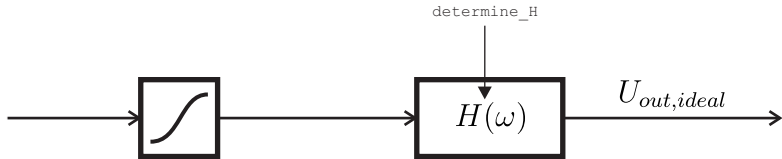


## Code: Die Bausteine



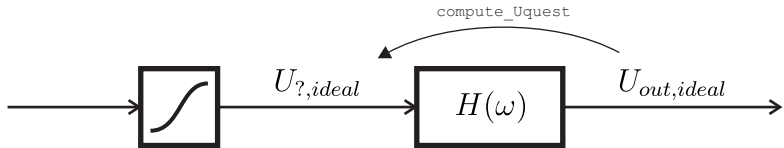
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )
```

## Code: Die Bausteine



```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )  
2 H = dertermine_H ( )
```

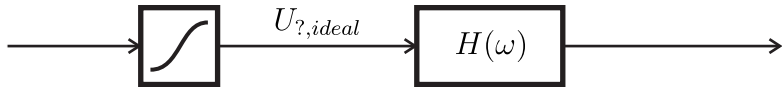
## Code: Die Bausteine



```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )  
2 H = dertermine_H ( )  
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )
```

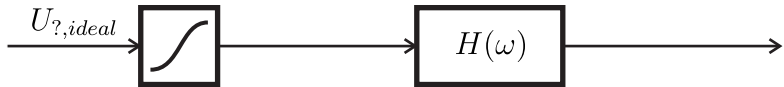


## Code: Die Bausteine



```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )  
2 H = dertermine_H ( )  
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )
```

## Code: Die Bausteine



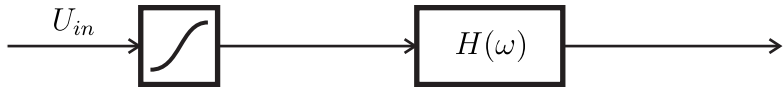
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )  
2 H = dertermine_H ( )  
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )
```

## Code: Die Bausteine



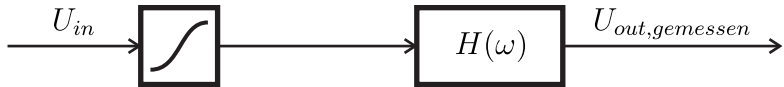
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )  
2 H = dertermine_H ( )  
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )  
4 Uin = Uquest_ideal
```

## Code: Die Bausteine



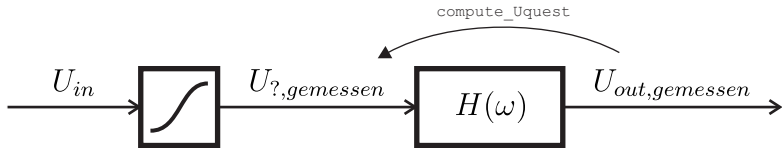
```
1 Uout_ideal = generate_BBsignal ( fq_rep, fq_bb, vpp )  
2 H = dertermine_H ( )  
3 Uquest_ideal = compute_Uquest ( Uout_ideal, H )  
4 Uin = Uquest_ideal  
5 Uout_measured = measure_Uout ( Uin )
```

## Code: Die Bausteine



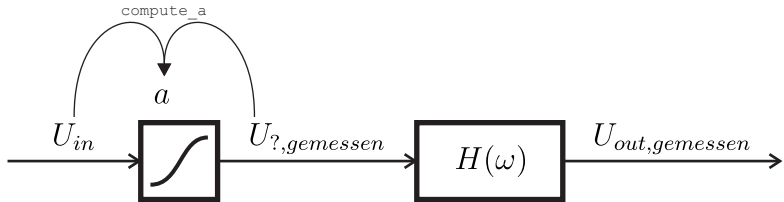
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )
2 H = dertermine_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )
4 Uin = Uquest_ideal
5 Uout_measured = measure_Uout ( Uin )
```

## Code: Die Bausteine



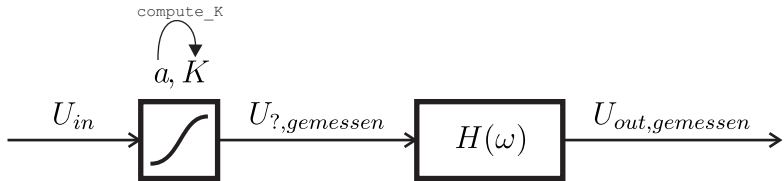
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )
2 H = dertermine_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )
4 Uin = Uquest_ideal
5 Uout_measured = measure_Uout ( Uin )
6 Uquest_measured = compute_Uquest ( Uout_measured , H )
```

# Code: Die Bausteine



```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )
2 H = dertermine_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )
4 Uin = Uquest_ideal
5 Uout_measured = measure_Uout ( Uin )
6 Uquest_measured = compute_Uquest ( Uout_measured , H )
7 a = compute_a ( Uin , Uquest_measured , N )
```

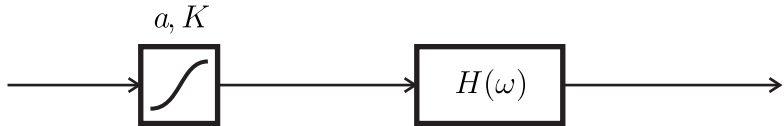
## Code: Die Bausteine



```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )
2 H = dertermine_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )
4 Uin = Uquest_ideal
5 Uout_measured = measure_Uout ( Uin )
6 Uquest_measured = compute_Uquest ( Uout_measured , H )
7 a = compute_a ( Uin , Uquest_measured , N )
8 K = compute_K ( a )
```

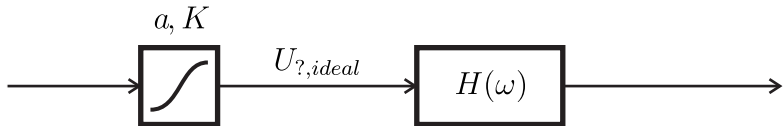


## Code: Die Bausteine



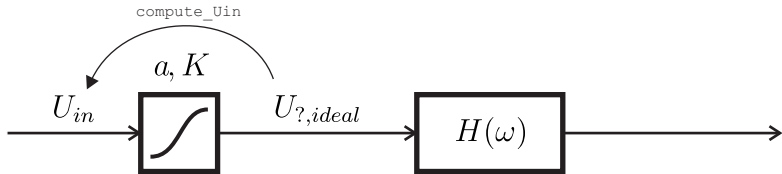
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )
2 H = dertermine_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )
4 Uin = Uquest_ideal
5 Uout_measured = measure_Uout ( Uin )
6 Uquest_measured = compute_Uquest ( Uout_measured , H )
7 a = compute_a ( Uin , Uquest_measured , N )
8 K = compute_K ( a )
```

## Code: Die Bausteine



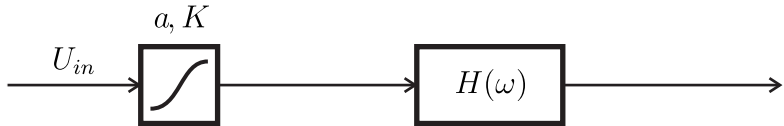
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )
2 H = dertermine_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )
4 Uin = Uquest_ideal
5 Uout_measured = measure_Uout ( Uin )
6 Uquest_measured = compute_Uquest ( Uout_measured , H )
7 a = compute_a ( Uin , Uquest_measured , N )
8 K = compute_K ( a )
```

## Code: Die Bausteine



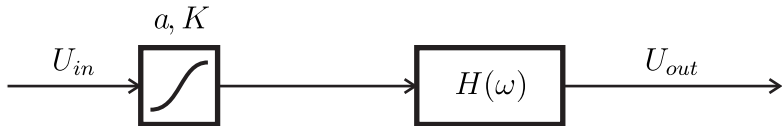
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )
2 H = dertermine_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )
4 Uin = Uquest_ideal
5 Uout_measured = measure_Uout ( Uin )
6 Uquest_measured = compute_Uquest ( Uout_measured , H )
7 a = compute_a ( Uin , Uquest_measured , N )
8 K = compute_K ( a )
9 Uin = compute_Uin ( Uquest_ideal , K )
```

## Code: Die Bausteine



```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )
2 H = dertermine_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )
4 Uin = Uquest_ideal
5 Uout_measured = measure_Uout ( Uin )
6 Uquest_measured = compute_Uquest ( Uout_measured , H )
7 a = compute_a ( Uin , Uquest_measured , N )
8 K = compute_K ( a )
9 Uin = compute_Uin ( Uquest_ideal , K )
```

## Code: Die Bausteine



```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )
2 H = dertermine_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )
4 Uin = Uquest_ideal
5 Uout_measured = measure_Uout ( Uin )
6 Uquest_measured = compute_Uquest ( Uout_measured , H )
7 a = compute_a ( Uin , Uquest_measured , N )
8 K = compute_K ( a )
9 Uin = compute_Uin ( Uquest_ideal , K )
10 Uout = measure_Uout ( Uin )
```



---

## Code: Vorgehensweise

---

---

## Code: Vorgehensweise

---

- Refactoring / Anpassung der Matlab-Funktionen an unser Design

## Code: Vorgehensweise

---

- Refactoring / Anpassung der Matlab-Funktionen an unser Design
- Portierung der Matlab-Funktionen nach Python



## Code: Vorgehensweise

---

- Refactoring / Anpassung der Matlab-Funktionen an unser Design
- Portierung der Matlab-Funktionen nach Python
- Überprüfung der portierten Funktionen mithilfe von **TDD**

## Code: Vorgehensweise

---

- Refactoring / Anpassung der Matlab-Funktionen an unser Design
- Portierung der Matlab-Funktionen nach Python
- Überprüfung der portierten Funktionen mithilfe von **TDD**
- Erweiterung der Blöcken mit `adjust_H` und `adjust_a`

## Code: Vorgehensweise

---

- Refactoring / Anpassung der Matlab-Funktionen an unser Design
- Portierung der Matlab-Funktionen nach Python
- Überprüfung der portierten Funktionen mithilfe von **TDD**
- Erweiterung der Blöcken mit `adjust_H` und `adjust_a`
- Neue Klassen `signale_class` und `transfer_function_class`

---

# Test Driven Development

---

---

# Test Driven Development

---

- 27 Unit Tests

# Test Driven Development

---

- 27 Unit Tests
- 4 System Tests

# Test Driven Development

---

- 27 Unit Tests
- 4 System Tests

Vorteile:

# Test Driven Development

---

- 27 Unit Tests
- 4 System Tests

Vorteile:

- Ermöglichen:
  - inkrementierende Code-Anpassungen



# Test Driven Development

---

- 27 Unit Tests
- 4 System Tests

Vorteile:

- Ermöglichen:
  - inkrementierende Code-Anpassungen
  - verteiltes Debuggen ohne den Messaufbau

# Test Driven Development

---

- 27 Unit Tests
- 4 System Tests

Vorteile:

- Ermöglichen:
  - inkrementierende Code-Anpassungen
  - verteiltes Debuggen ohne den Messaufbau
- Zwingen zum modularen Code-Design

# Test Driven Development

---

- 27 Unit Tests
- 4 System Tests

Vorteile:

- Ermöglichen:
  - inkrementierende Code-Anpassungen
  - verteiltes Debuggen ohne den Messaufbau
- Zwingen zum modularen Code-Design
- Erleichtern das Migrieren der Funktionen aus anderen Sprachen

# Test Driven Development

---

- 27 Unit Tests
- 4 System Tests

Vorteile:

- Ermöglichen:
  - inkrementierende Code-Anpassungen
  - verteiltes Debuggen ohne den Messaufbau
- Zwingen zum modularen Code-Design
- Erleichtern das Migrieren der Funktionen aus anderen Sprachen
- Dienen auch als Dokumentation der Bausteine

# Test Driven Development

---

- 27 Unit Tests
- 4 System Tests

## Vorteile:

- Ermöglichen:
  - inkrementierende Code-Anpassungen
  - verteiltes Debuggen ohne den Messaufbau
- Zwingen zum modularen Code-Design
- Erleichtern das Migrieren der Funktionen aus anderen Sprachen
- Dienen auch als Dokumentation der Bausteine

## Nachteile:

# Test Driven Development

---

- 27 Unit Tests
- 4 System Tests

## Vorteile:

- Ermöglichen:
  - inkrementierende Code-Anpassungen
  - verteiltes Debuggen ohne den Messaufbau
- Zwingen zum modularen Code-Design
- Erleichtern das Migrieren der Funktionen aus anderen Sprachen
- Dienen auch als Dokumentation der Bausteine

## Nachteile:

- Extra Aufwand: mehr Code zu debuggen

# Test Driven Development

---

- 27 Unit Tests
- 4 System Tests

## Vorteile:

- Ermöglichen:
  - inkrementierende Code-Anpassungen
  - verteiltes Debuggen ohne den Messaufbau
- Zwingen zum modularen Code-Design
- Erleichtern das Migrieren der Funktionen aus anderen Sprachen
- Dienen auch als Dokumentation der Bausteine

## Nachteile:

- Extra Aufwand: mehr Code zu debuggen
- Gewöhnungsbedürftig

---

# Das Mock-System

---



# Das Mock-System

---

- Wird genutzt, wenn mit Geräten kommuniziert wird:
  - `mock_system.write_to_AWG`
  - `mock_system.read_from_DSO`

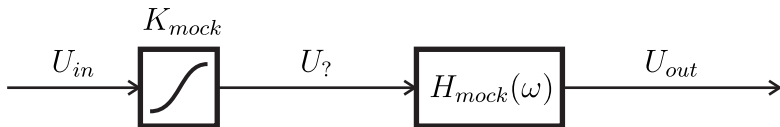
# Das Mock-System

---

- Wird genutzt, wenn mit Geräten kommuniziert wird:
  - `mock_system.write_to_AWG`
  - `mock_system.read_from_DSO`
- Simuliert das Verhalten des Messaufbaus nach dem Hammerstein Model

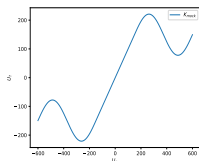
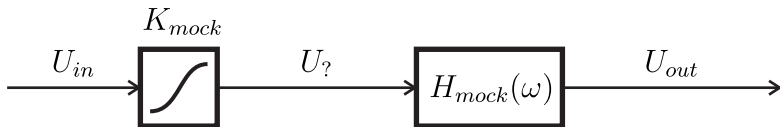
# Das Mock-System

- Wird genutzt, wenn mit Geräten kommuniziert wird:
  - `mock_system.write_to_AWG`
  - `mock_system.read_from_DSO`
- Simuliert das Verhalten des Messaufbaus nach dem Hammerstein Model



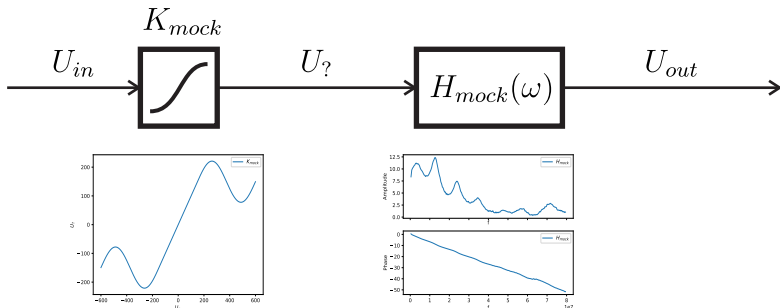
# Das Mock-System

- Wird genutzt, wenn mit Geräten kommuniziert wird:
  - `mock_system.write_to_AWG`
  - `mock_system.read_from_DSO`
- Simuliert das Verhalten des Messaufbaus nach dem Hammerstein Model



# Das Mock-System

- Wird genutzt, wenn mit Geräten kommuniziert wird:
  - `mock_system.write_to_AWG`
  - `mock_system.read_from_DSO`
- Simuliert das Verhalten des Messaufbaus nach dem Hammerstein Model



# Das Mock-System

---

- Wird genutzt, wenn mit Geräten kommuniziert wird:
  - `mock_system.write_to_AWG`
  - `mock_system.read_from_DSO`
- Simuliert das Verhalten des Messaufbaus nach dem Hammerstein Model

Vorteile:

# Das Mock-System

---

- Wird genutzt, wenn mit Geräten kommuniziert wird:
  - `mock_system.write_to_AWG`
  - `mock_system.read_from_DSO`
- Simuliert das Verhalten des Messaufbaus nach dem Hammerstein Model

Vorteile:

- Ermöglicht:
  - Unit Tests von Bausteinen, in den Gerätekommunikation stattfindet

# Das Mock-System

---

- Wird genutzt, wenn mit Geräten kommuniziert wird:
  - `mock_system.write_to_AWG`
  - `mock_system.read_from_DSO`
- Simuliert das Verhalten des Messaufbaus nach dem Hammerstein Model

## Vorteile:

- Ermöglicht:
  - Unit Tests von Bausteinen, in den Gerätekommunikation stattfindet
  - System Tests



# Das Mock-System

---

- Wird genutzt, wenn mit Geräten kommuniziert wird:
  - `mock_system.write_to_AWG`
  - `mock_system.read_from_DSO`
- Simuliert das Verhalten des Messaufbaus nach dem Hammerstein Model

## Vorteile:

- Ermöglicht:
  - Unit Tests von Bausteinen, in den Gerätekommunikation stattfindet
  - System Tests
  - Testen von Randfällen

# Das Mock-System

---

- Wird genutzt, wenn mit Geräten kommuniziert wird:
  - `mock_system.write_to_AWG`
  - `mock_system.read_from_DSO`
- Simuliert das Verhalten des Messaufbaus nach dem Hammerstein Model

## Vorteile:

- Ermöglicht:
  - Unit Tests von Bausteinen, in den Gerätekommunikation stattfindet
  - System Tests
  - Testen von Randfällen
- Hilft das System besser zu verstehen

# Das Mock-System

---

- Wird genutzt, wenn mit Geräten kommuniziert wird:
  - `mock_system.write_to_AWG`
  - `mock_system.read_from_DSO`
- Simuliert das Verhalten des Messaufbaus nach dem Hammerstein Model

## Vorteile:

- Ermöglicht:
  - Unit Tests von Bausteinen, in den Gerätekommunikation stattfindet
  - System Tests
  - Testen von Randfällen
- Hilft das System besser zu verstehen

## Nachteile:

- Extra Aufwand: mehr Code zu debuggen

# Empfehlungen zum Code-Design

---

- Weiter Klasse `K_class` implementieren

# Empfehlungen zum Code-Design

---

- Weiter Klasse `K_class` implementieren
- Refactoring



---

# Überlegungen

---

# Überlegungen

---

- Reihenfolge der Optimierung: Parallele Iteration  $\Leftrightarrow$  alternierende Iteration von  $H$  und  $K$

# Überlegungen

---

- Reihenfolge der Optimierung: Parallele Iteration  $\Leftrightarrow$  alternierende Iteration von  $H$  und  $K$
- Einfluss von  $K$  auf das Spektrum von  $U_?$  und damit auf Optimierung von  $H$  durch Oberschwingungen bei Potenzierung des Eingangssignals



# Überlegungen

---

- Reihenfolge der Optimierung: Parallele Iteration  $\Leftrightarrow$  alternierende Iteration von  $H$  und  $K$
- Einfluss von  $K$  auf das Spektrum von  $U_?$  und damit auf Optimierung von  $H$  durch Oberschwingungen bei Potenzierung des Eingangssignals
- Auswahl der Schrittweiten in den Optimierungsalgorithmen

# Überlegungen

---

- Reihenfolge der Optimierung: Parallele Iteration  $\Leftrightarrow$  alternierende Iteration von  $H$  und  $K$
- Einfluss von  $K$  auf das Spektrum von  $U_?$  und damit auf Optimierung von  $H$  durch Oberschwingungen bei Potenzierung des Eingangssignals
- Auswahl der Schrittweiten in den Optimierungsalgorithmen
- Umgang mit Nulldurchgängen des idealen Spektrums in Optimierung von  $H$

# Überlegungen

---

- Reihenfolge der Optimierung: Parallele Iteration  $\Leftrightarrow$  alternierende Iteration von  $H$  und  $K$
- Einfluss von  $K$  auf das Spektrum von  $U_?$  und damit auf Optimierung von  $H$  durch Oberschwingungen bei Potenzierung des Eingangssignals
- Auswahl der Schrittweiten in den Optimierungsalgorithmen
- Umgang mit Nulldurchgängen des idealen Spektrums in Optimierung von  $H$
- Einfluss von Rauschen auf Optimierungsalgorithmen

# Überlegungen

---

- Reihenfolge der Optimierung: Parallele Iteration  $\Leftrightarrow$  alternierende Iteration von  $H$  und  $K$
- Einfluss von  $K$  auf das Spektrum von  $U_?$  und damit auf Optimierung von  $H$  durch Oberschwingungen bei Potenzierung des Eingangssignals
- Auswahl der Schrittweiten in den Optimierungsalgorithmen
- Umgang mit Nulldurchgängen des idealen Spektrums in Optimierung von  $H$
- Einfluss von Rauschen auf Optimierungsalgorithmen
- Anpassung der Phase in Optimierung von  $H$

# Überlegungen

---

- Reihenfolge der Optimierung: Parallele Iteration  $\Leftrightarrow$  alternierende Iteration von  $H$  und  $K$
- Einfluss von  $K$  auf das Spektrum von  $U_?$  und damit auf Optimierung von  $H$  durch Oberschwingungen bei Potenzierung des Eingangssignals
- Auswahl der Schrittweiten in den Optimierungsalgorithmen
- Umgang mit Nulldurchgängen des idealen Spektrums in Optimierung von  $H$
- Einfluss von Rauschen auf Optimierungsalgorithmen
- Anpassung der Phase in Optimierung von  $H$
- Definitionsbereich von  $K$  bei initialer Berechnung und Optimierung