

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
 - Problemstellung
 - Aufbau
- 2** Design
 - Blöcke
 - Test Driven Development
 - MockSystem
- 3** Optimierung
 - Optimierung der Übertragungsfunktion
 - Optimierung der Kennlinie

Problemstellung

- Barrier-Bucket System

Problemstellung

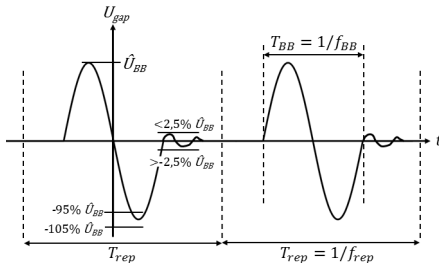
- Barrier-Bucket System :
 - Longitudinale Manipulation des Teilchenstrahls

Problemstellung

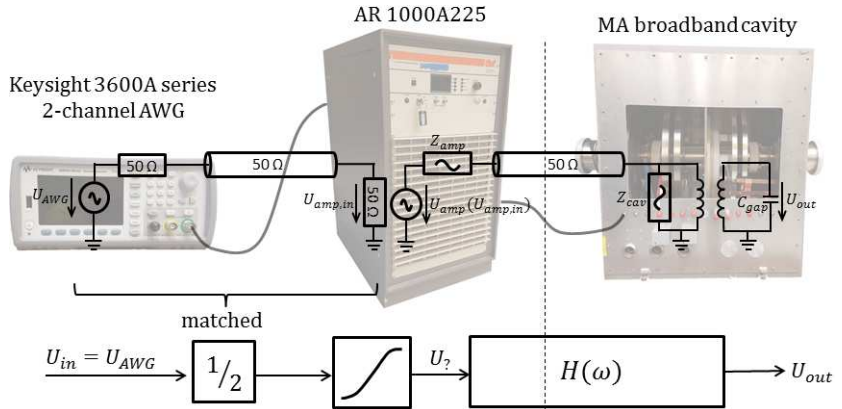
- Barrier-Bucket System :
 - Longitudinale Manipulation des Teilchenstrahls
- Ziel

Problemstellung

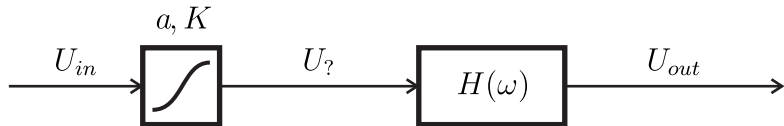
- Barrier-Bucket System :
 - Longitudinale Manipulation des Teilchenstrahls
- Ziel :
 - Gap Spannung in Form einer Ein-Sinus Periode
 - Qualität das Signals



Aufbau und Modell

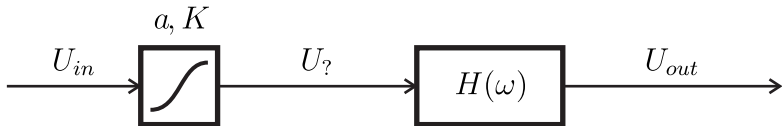


Aufbau und Modell



Aufbau und Modell

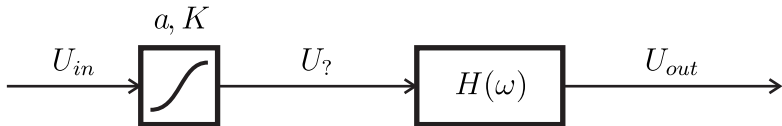
- Gegeben:
 - Lineare Übertragungsfunktion H bestimmt durch Pseudorauschen
 - System linear bis $\hat{U}_{BB} \approx 550 \text{ V}$ genähert



Aufbau und Modell

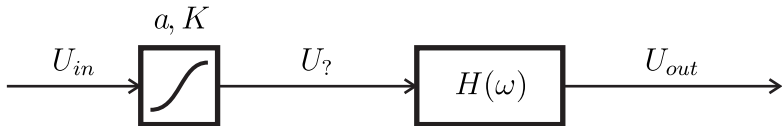
- Gegeben:
 - Lineare Übertragungsfunktion H bestimmt durch Pseudorauschen
 - System linear bis $\hat{U}_{BB} \approx 550 \text{ V}$ genähert
- Hammerstein Modell :

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



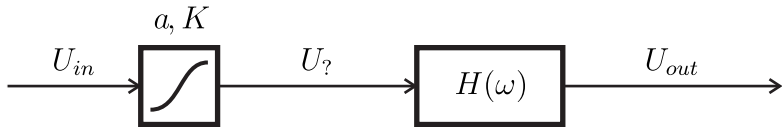
Aufbau und Modell

- 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
$$U_{\gamma}(t) = \sum_{n=1}^N a_n [U_{in}(t)]^n \quad \underline{U}_{out}(\omega) = H(\omega) \cdot \underline{U}_{\gamma}(\omega)$$
- Zielsetzung :
 - Parameter a_n der Kennlinie K zu bestimmen
 - Ersten Optimierungs Ansatz implementieren

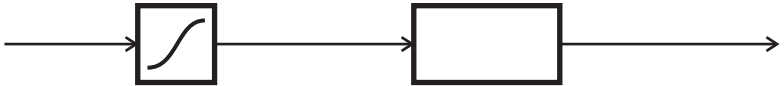


«««< HEAD

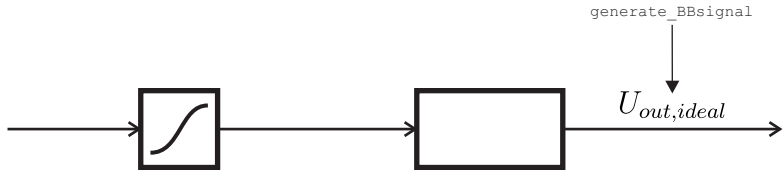
Code: Die Blöcke



Code: Die Blöcke

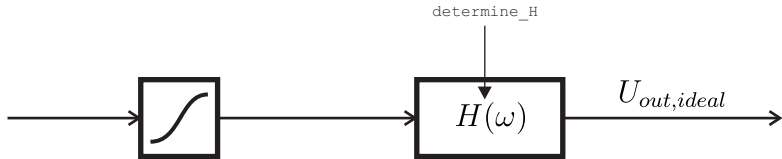


Code: Die Blöcke



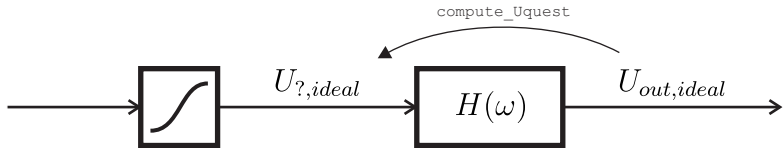
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )
```

Code: Die Blöcke



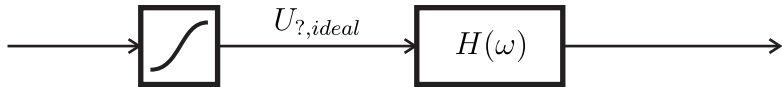
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )  
2 H = determine_H ( )
```


Code: Die Blöcke



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

Code: Die Blöcke



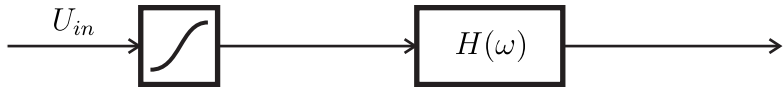
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )  
2 H = determine_H ( )  
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )
```

Code: Die Blöcke



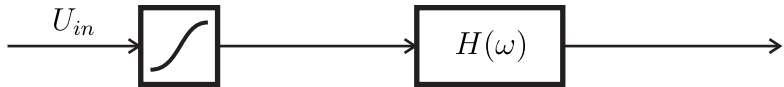
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )  
2 H = determine_H ( )  
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )
```

Code: Die Blöcke



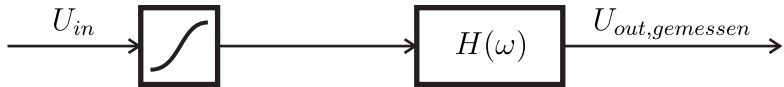
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )  
2 H = determine_H ( )  
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )  
4 Uin = Uquest_ideal
```

Code: Die Blöcke



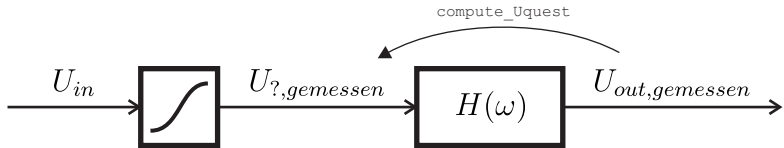
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )  
2 H = determine_H ( )  
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )  
4 Uin = Uquest_ideal  
5 Uout_measured = measure_Uout ( Uin )
```

Code: Die Blöcke



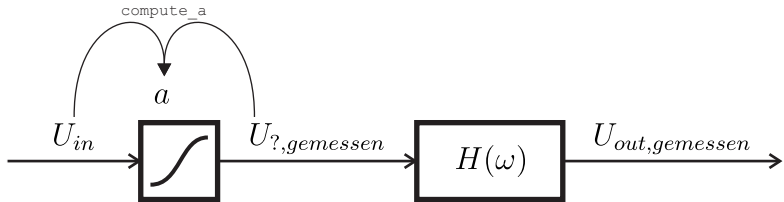
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )  
2 H = determine_H ( )  
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )  
4 Uin = Uquest_ideal  
5 Uout_measured = measure_Uout ( Uin )
```

Code: Die Blöcke



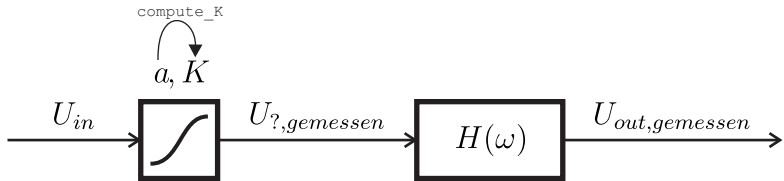
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )
2 H = determine_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )
4 Uin = Uquest_ideal
5 Uout_measured = measure_Uout ( Uin )
6 Uquest_measured = compute_Uquest ( Uout_measured , H )
```

Code: Die Blöcke



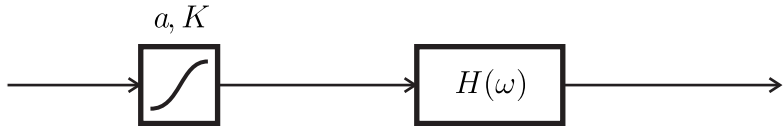
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )
2 H = determine_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )
4 Uin = Uquest_ideal
5 Uout_measured = measure_Uout ( Uin )
6 Uquest_measured = compute_Uquest ( Uout_measured , H )
7 a = compute_a ( Uin , Uquest_measured , N )
```


Code: Die Blöcke



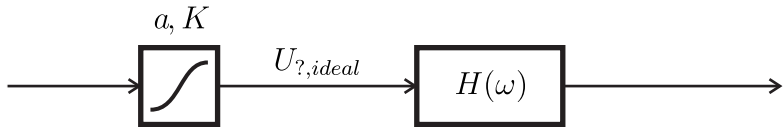
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )
2 H = determine_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )
4 Uin = Uquest_ideal
5 Uout_measured = measure_Uout ( Uin )
6 Uquest_measured = compute_Uquest ( Uout_measured , H )
7 a = compute_a ( Uin , Uquest_measured , N )
8 K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

Code: Die Blöcke



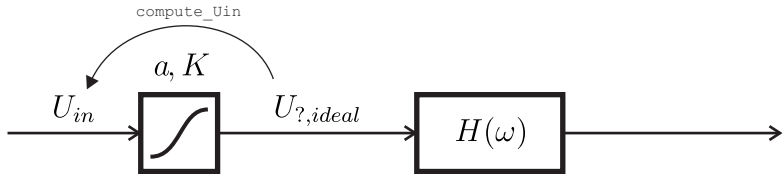
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )
2 H = determine_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )
4 Uin = Uquest_ideal
5 Uout_measured = measure_Uout ( Uin )
6 Uquest_measured = compute_Uquest ( Uout_measured , H )
7 a = compute_a ( Uin , Uquest_measured , N )
8 K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

Code: Die Blöcke



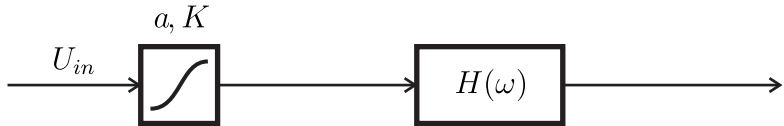
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )
2 H = determine_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )
4 Uin = Uquest_ideal
5 Uout_measured = measure_Uout ( Uin )
6 Uquest_measured = compute_Uquest ( Uout_measured , H )
7 a = compute_a ( Uin , Uquest_measured , N )
8 K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

Code: Die Blöcke



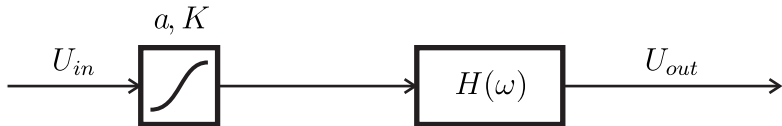
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )
2 H = determine_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )
4 Uin = Uquest_ideal
5 Uout_measured = measure_Uout ( Uin )
6 Uquest_measured = compute_Uquest ( Uout_measured , H )
7 a = compute_a ( Uin , Uquest_measured , N )
8 K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

Code: Die Blöcke



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



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

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

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

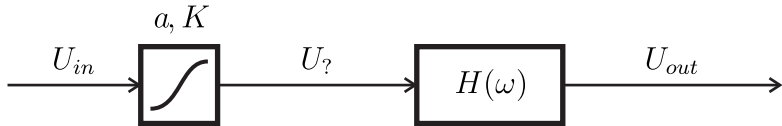
Nachteile:

- Extra Aufwand: mehr Code zu debuggen

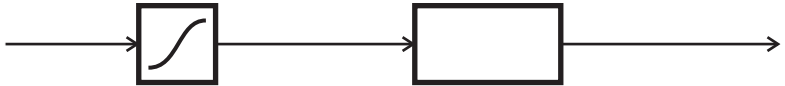


=====

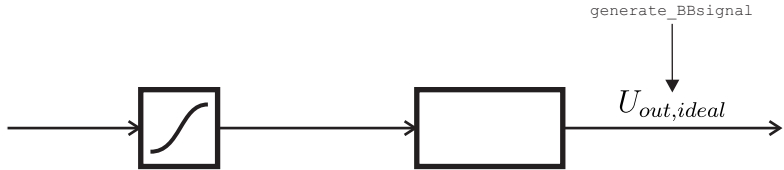
Code: Die Blöcke



Code: Die Blöcke

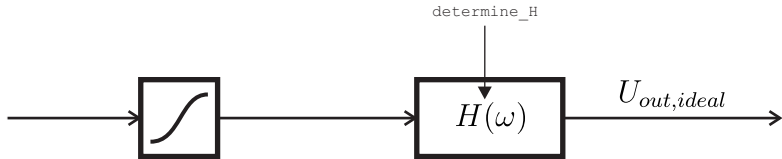


Code: Die Blöcke



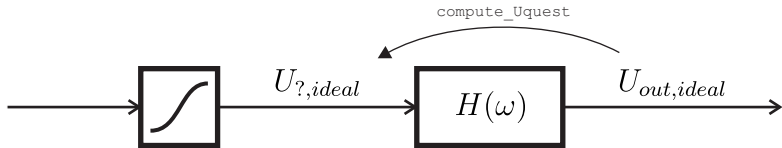
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )
```

Code: Die Blöcke



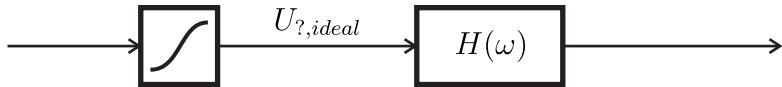
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )  
2 H = determine_H ( )
```

Code: Die Blöcke



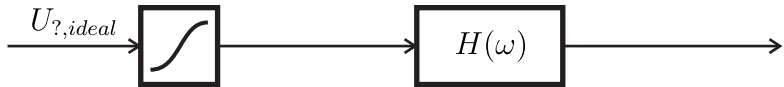
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )  
2 H = determine_H ( )  
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )
```

Code: Die Blöcke



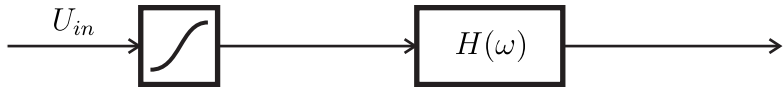
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )  
2 H = determine_H ( )  
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )
```

Code: Die Blöcke



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


Code: Die Blöcke



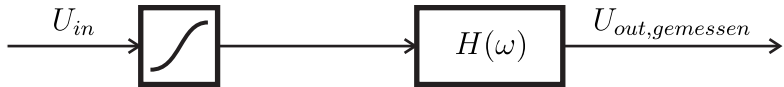
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )  
2 H = determine_H ( )  
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )  
4 Uin = Uquest_ideal
```

Code: Die Blöcke



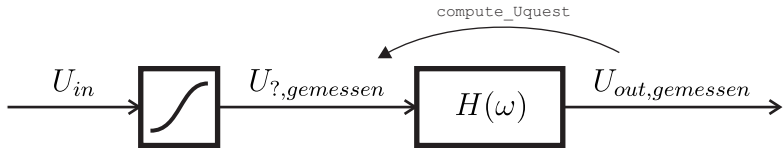
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )  
2 H = determine_H ( )  
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )  
4 Uin = Uquest_ideal  
5 Uout_measured = measure_Uout ( Uin )
```

Code: Die Blöcke



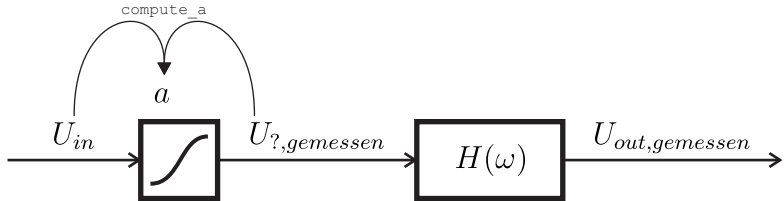
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )
2 H = determine_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )
4 Uin = Uquest_ideal
5 Uout_measured = measure_Uout ( Uin )
```

Code: Die Blöcke



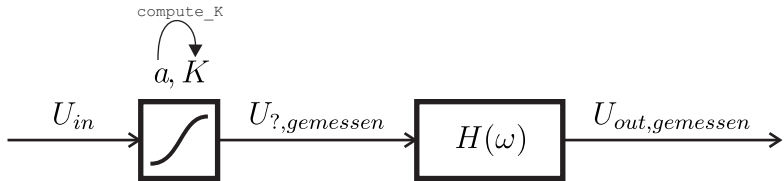
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )
2 H = determine_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )
4 Uin = Uquest_ideal
5 Uout_measured = measure_Uout ( Uin )
6 Uquest_measured = compute_Uquest ( Uout_measured , H )
```

Code: Die Blöcke



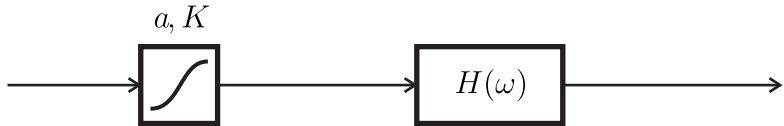
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )
2 H = determine_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )
4 Uin = Uquest_ideal
5 Uout_measured = measure_Uout ( Uin )
6 Uquest_measured = compute_Uquest ( Uout_measured , H )
7 a = compute_a ( Uin , Uquest_measured , N )
```

Code: Die Blöcke



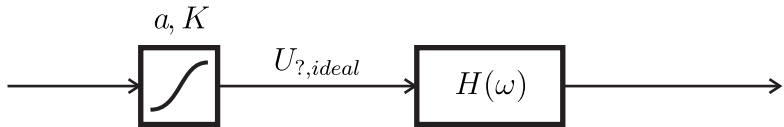
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )
2 H = determine_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )
4 Uin = Uquest_ideal
5 Uout_measured = measure_Uout ( Uin )
6 Uquest_measured = compute_Uquest ( Uout_measured , H )
7 a = compute_a ( Uin , Uquest_measured , N )
8 K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

Code: Die Blöcke



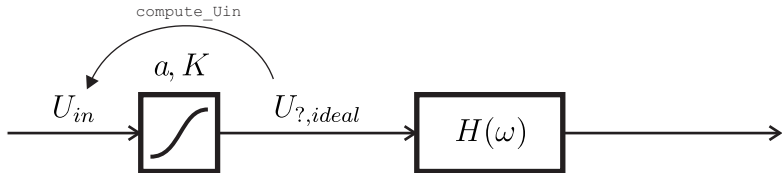
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )  
2 H = determine_H ( )  
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )  
4 Uin = Uquest_ideal  
5 Uout_measured = measure_Uout ( Uin )  
6 Uquest_measured = compute_Uquest ( Uout_measured , H )  
7 a = compute_a ( Uin , Uquest_measured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

Code: Die Blöcke



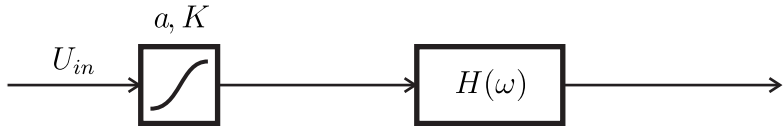
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )
2 H = determine_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )
4 Uin = Uquest_ideal
5 Uout_measured = measure_Uout ( Uin )
6 Uquest_measured = compute_Uquest ( Uout_measured , H )
7 a = compute_a ( Uin , Uquest_measured , N )
8 K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


Code: Die Blöcke



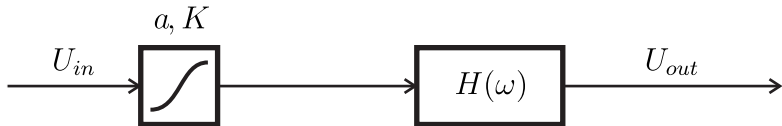
```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )
2 H = determine_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal , H )
4 Uin = Uquest_ideal
5 Uout_measured = measure_Uout ( Uin )
6 Uquest_measured = compute_Uquest ( Uout_measured , H )
7 a = compute_a ( Uin , Uquest_measured , N )
8 K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

Code: Die Blöcke



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



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



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

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

Nachteile:

- Extra Aufwand: mehr Code zu debuggen



»»»> b169fb0089a2e9f292516d071d31fc35f2a151dd

Das Mock-System

Das Mock-System

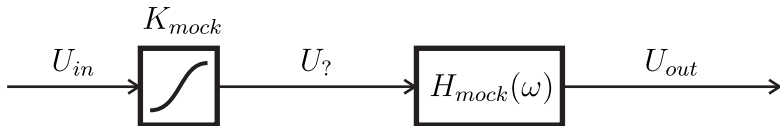
- Wird genutzt, wenn mit Geräten kommuniziert wird:
 - `mock_system.write_to_AWG`
 - `mock_system.read_from_DSO`

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

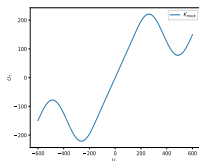
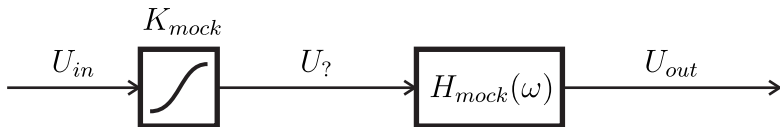
Das Mock-System

- 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



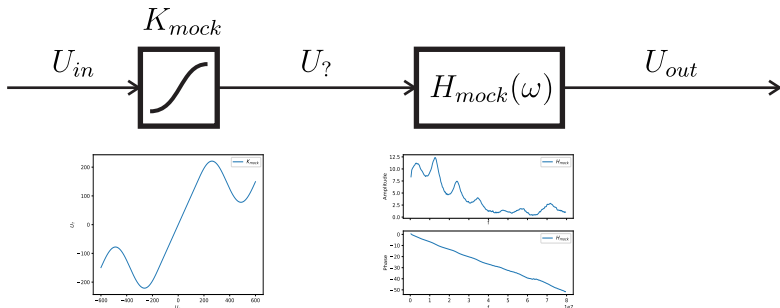
Das Mock-System

- 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 Spezialszenarien

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 Spezialszenarien
- 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 Spezialszenarien
- Hilft das System besser zu verstehen

Nachteile:

- Extra Aufwand: mehr Code zu debuggen



Optimierung der Übertragungsfunktion

Optimierung der Übertragungsfunktion

Idee: iterative Anpassung mit

$$\underline{H}^{i+1}(\omega) = \underline{H}^i(\omega) \left(1 + \sigma_H \cdot \left(\frac{\underline{U}_{out,mess}^i(\omega)}{\underline{U}_{out,ideal}^i(\omega)} - 1 \right) \right)$$

mit σ_H als Schrittweite.

Optimierung der Übertragungsfunktion

Idee: iterative Anpassung mit

$$\underline{H}^{i+1}(\omega) = \underline{H}^i(\omega) \left(1 + \sigma_H \cdot \left(\frac{\underline{U}_{out,mess}^i(\omega)}{\underline{U}_{out,ideal}^i(\omega)} - 1 \right) \right)$$

mit σ_H als Schrittweite.

- Fokus auf Optimierung des Betrags nach ersten Messungen mit Phasenanpassung

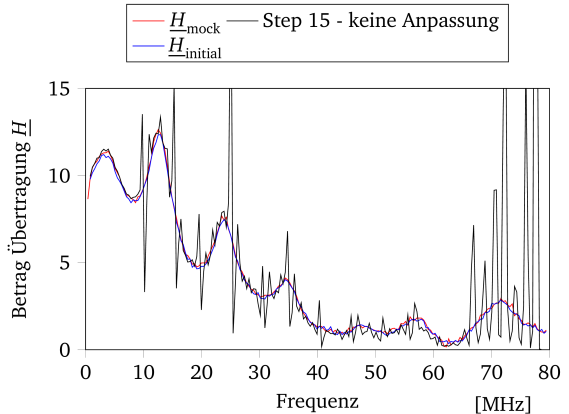
Optimierung der Übertragungsfunktion

Idee: iterative Anpassung mit

$$\underline{H}^{i+1}(\omega) = \underline{H}^i(\omega)$$

mit σ_H als Schrittweite.

- Fokus auf Optimierung d Phasenanpassung



Optimierung der Übertragungsfunktion

Idee: iterative Anpassung mit

$$\underline{H}^{i+1}(\omega) = \underline{H}^i(\omega) \left(1 + \sigma_H \cdot \left(\frac{\underline{U}_{out,mess}^i(\omega)}{\underline{U}_{out,ideal}^i(\omega)} - 1 \right) \right)$$

mit σ_H als Schrittweite.

- Fokus auf Optimierung des Betrags nach ersten Messungen mit Phasenanpassung
- Umgang mit Fehlereinflüssen?



Optimierung der Übertragungsfunktion

Optimierung der Übertragungsfunktion

Fehlerquellen:

Optimierung der Übertragungsfunktion

Fehlerquellen:

- Diskretisierungsfehler durch die FFT

Optimierung der Übertragungsfunktion

Fehlerquellen:

- Diskretisierungsfehler durch die FFT
- Interpolationsfehler bei Auswertung des Korrektur-Terms

Optimierung der Übertragungsfunktion

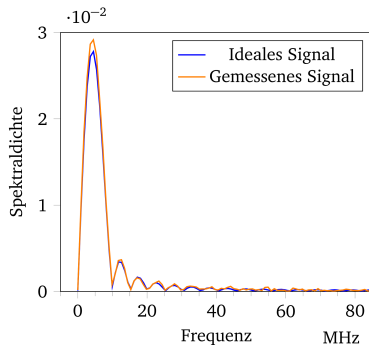
Fehlerquellen:

- Diskretisierungsfehler durch die FFT
- Interpolationsfehler bei Auswertung des Korrektur-Terms
- Rauscheinflüsse bei Messungen

Optimierung der Übertragungsfunktion

Fehlerquellen:

- Diskretisierungsfehler durch die FFT
- Interpolationsfehler bei Auswertung des Korrektur-Terms
- Rauscheinflüsse bei Messungen



Optimierung der Übertragungsfunktion

Fehlerquellen:

- Diskretisierungsfehler durch die FFT
- Interpolationsfehler bei Auswertung des Korrektur-Terms
- Rauscheinflüsse bei Messungen

Erste Lösungsansätze:

Optimierung der Übertragungsfunktion

Fehlerquellen:

- Diskretisierungsfehler durch die FFT
- Interpolationsfehler bei Auswertung des Korrektur-Terms
- Rauscheinflüsse bei Messungen

Erste Lösungsansätze:

- Ignorieren kleiner Beträge in Spektren der Signale

Optimierung der Übertragungsfunktion

Fehlerquellen:

- Diskretisierungsfehler durch die FFT
- Interpolationsfehler bei Auswertung des Korrektur-Terms
- Rauscheinflüsse bei Messungen

Erste Lösungsansätze:

- Ignorieren kleiner Beträge in Spektren der Signale
- Ignorieren großer Korrektur-Terme

Optimierung der Übertragungsfunktion

Fehlerquellen:

- Diskretisierungsfehler durch die FFT
- Interpolationsfehler bei Auswertung des Korrektur-Terms
- Rauscheinflüsse bei Messungen

Erste Lösungsansätze:

- Ignorieren kleiner Beträge in Spektren der Signale
- Ignorieren großer Korrektur-Terme
- *Zero-Padding*

Optimierung der Übertragungsfunktion

Fehlerquellen:

- Diskretisierungsfehler durch die FFT
- Interpolationsfehler bei Auswertung des Korrektur-Terms
- Rauscheinflüsse bei Messungen

Erste Lösungsansätze:

- Ignorieren kleiner Beträge in Spektren der Signale
- Ignorieren großer Korrektur-Terme
- *Zero-Padding*

—> Simulation der komplexen Optimierung am Mock-System

Optimierung der Übertragungsfunktion

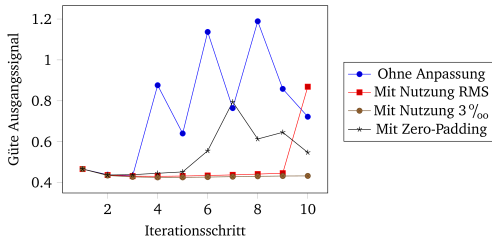
Fehlerquellen:

- Diskretisierungsfehler durch die
- Interpolationsfehler bei Auswertung
- Rauscheinflüsse bei Messungen

Erste Lösungsansätze:

- Ignorieren kleiner Beträge in Spektren der Signale
- Ignorieren großer Korrektur-Terme
- *Zero-Padding*

→ Simulation der komplexen Optimierung am Mock-System



Optimierung der Übertragungsfunktion

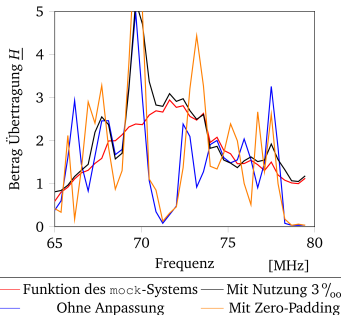
Fehlerquellen:

- Diskretisierungsfehler durch die FFT
- Interpolationsfehler bei Auswertung de
- Rauscheinflüsse bei Messungen

Erste Lösungsansätze:

- Ignorieren kleiner Beträge in Spektren
- Ignorieren großer Korrektur-Terme
- *Zero-Padding*

—> Simulation der komplexen Optimierung am Mock-System



Optimierung der Übertragungsfunktion

Fehlerquellen:

- Diskretisierungsfehler durch die FFT
- Interpolationsfehler bei Auswertung des Korrektur-Terms
- Rauscheinflüsse bei Messungen

Erste Lösungsansätze:

- Ignorieren kleiner Beträge in Spektren der Signale
- Ignorieren großer Korrektur-Terme
- *Zero-Padding*

—> Simulation der komplexen Optimierung am Mock-System

Ergebnis: noch nicht ausgereift

Optimierung von K

- Eine Kennlinie an das momentane Signal anpassen

$$U_{?,\text{meas}}(t) = \sum_{n=1}^N \bar{a}_n [U_{in}(t)]^n \quad U_{?,\text{ideal}}(t) = \sum_{n=1}^N a_n [U_{in}(t)]^n \quad (1)$$

- Oder direkt über die Differenz der Signale

$$\Delta U_?(t) = U_{?,\text{meas}}(t) - U_{?,\text{ideal}}(t) = \sum_{n=1}^N (\bar{a}_n - a_n) [U_{in}(t)]^n = \sum_{n=1}^N \tilde{a}_n [U_{in}(t)]^n \quad (2)$$

Optimierung von K

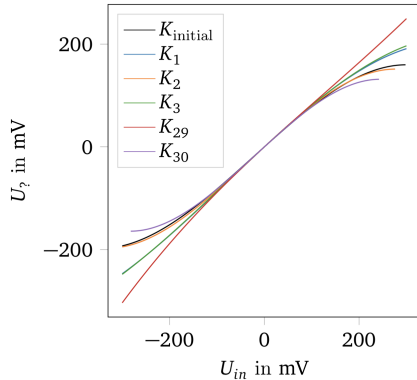
- Bestimmung der Parameter \tilde{a}_n
- Vergleichen der Samples $\Delta U_{?,i} = \Delta U_{?}(i \cdot \Delta t)$ mit $U_{in,i} = U_{in}(i \cdot \Delta t)$

$$\begin{pmatrix} U_{in,1} & U_{in,1}^2 & \dots & U_{in,1}^N \\ U_{in,2} & U_{in,2}^2 & \dots & U_{in,2}^N \\ \vdots & \vdots & \ddots & \vdots \\ U_{in,M} & U_{in,M}^2 & \dots & U_{in,M}^N \end{pmatrix} \cdot \begin{pmatrix} \tilde{a}_1 \\ \tilde{a}_2 \\ \vdots \\ \tilde{a}_N \end{pmatrix} = \begin{pmatrix} \Delta U_{?,1} \\ \Delta U_{?,2} \\ \vdots \\ \Delta U_{?,M} \end{pmatrix} \quad (3)$$

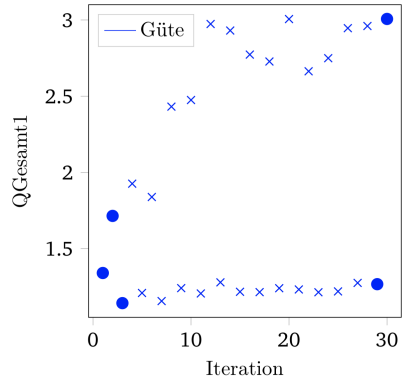
- Lösung des linearen Optimierungsproblems ergibt die Anpassung der alten Parameter

$$a_n^{i+1} = a_n^i + \sigma_a^i \tilde{a}_n^i \quad (4)$$

Erster Ansatz

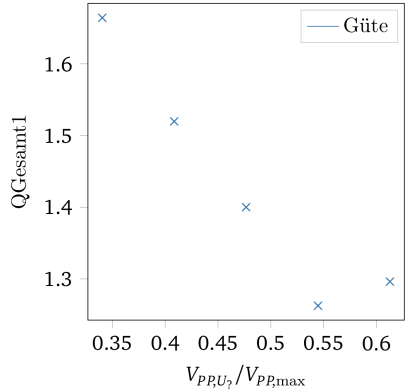
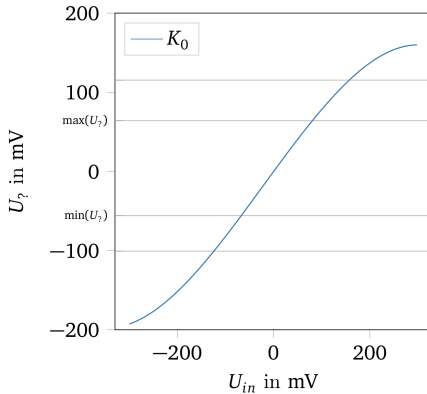


(a) Kennlinien



(b) Qualität

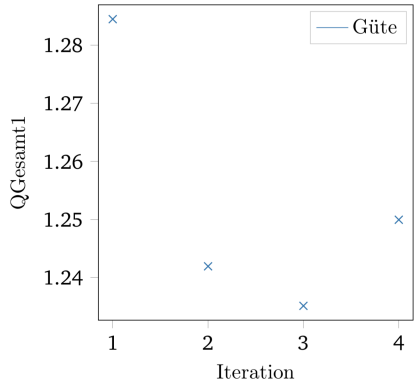
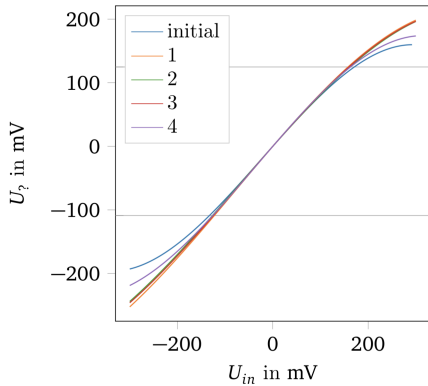
Grenzen der Kennlinie



Zweiter Ansatz

- K in einem kleineren Spannungsbereich anpassen
- Es wurden 66% des maximal zulässigen Bereichs verwendet

Zweiter Ansatz



Offene Punkte

- Zweiter Ansatz mit mehr Iterationen Testen
- Die Grenzen für K allgemein bestimmen

Ausblick

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
- 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
- Umgang mit Nulldurchgängen des idealen Spektrums in Optimierung von H
- 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
- Umgang mit Nulldurchgängen des idealen Spektrums in Optimierung von H
- Auswahl der Schrittweiten in den Optimierungsalgorithmen
- 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
- Umgang mit Nulldurchgängen des idealen Spektrums in Optimierung von H
- Auswahl der Schrittweiten in den Optimierungsalgorithmen
- Einfluss von Rauschen auf Optimierungsalgorithmen
- Definitionsbereich von K bei initialer Berechnung und Optimierung

Quellen

- Denys Bast, Armin Galetzka, "Projektseminar Beschleunigertechnik", 2017
- Jens Harzheim *et al.*, "Input Signal Generation For Barrier Bucket RF Systems At GSI",
- Jens Harzheim, "Idee iterative Optimierung der BB-Vorverzerrung" 2018.
- Kerstin Gross *et al.*, "Test Setup For Automated Barrier Bucket Signal Generation", 2017
- Julius Smith, "Mathematics of the Discrete Fourier Transform (DFT), Second Edition" W3K Publishing, 2007.
- Ian Sommerville, "Software Engineering, 9th edition", Pearson, 2012.