



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Kapitel 03

GRENZWERTANALYSE &
PYTEST.APPROX()

Lernziele heute

Am Ende dieser Vorlesung können Sie:

-  **Grenzwerte systematisch identifizieren** für numerische Funktionen
-  **IEEE 754-Grenzwerte nutzen** in Ihren Tests (`sys.float_info` , `math.inf` , `math.nan`)
-  **Korrekte Fließkomma-Assertions schreiben** mit `pytest.approx()`
-  **White-Box-Grenzwerttests anwenden** um Randfälle zu finden

Fokus: Tests schreiben, die echte Bugs fangen, nicht nur Happy-Path-Szenarien!

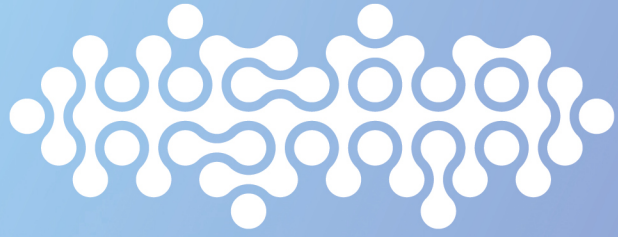
Warum Grenzwerte wichtig sind

Beobachtung: Die meisten Bugs verstecken sich an den **Rändern** von Äquivalenzklassen.

Typische Grenzwert-Bugs:

- ✗ Off-by-One-Fehler (`<=` vs `<`)
- ✗ Fließkomma-Präzisionsprobleme
- ✗ Überlauf/Unterlauf an numerischen Grenzen
- ✗ Division durch Null
- ✗ Unbehandelte Spezialwerte (NaN, Unendlich)

Beispiel: Ihre Tests bestehen für `reciprocal(5.0)`, aber schlagen fehl für `reciprocal(sys.float_info.max)` ✨



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

IEEE 754 & Float- Grenzwerte

1

IEEE 754: Die Grundlage des Float-Testens

Was ist IEEE 754?

Internationaler Standard für Fließkomma-Arithmetik (alle modernen Sprachen folgen ihm)

Warum das für Tests wichtig ist:

IEEE 754-Grenzwerte verstehen → Tests für Randfälle an den Grenzen numerischer Präzision schreiben

Python-Floats: 64-Bit IEEE 754 Doubles (wie C/C++ `double`)

Wichtige Erkenntnis: Es gibt **spezifische Grenzwerte**, die vom Standard definiert sind!

Kritische Float-Grenzwerte in Python

```
# ENDLICHE Grenzwerte
sys.float_info.max      #  $\approx 1.8e308$  - Größter endlicher Float
sys.float_info.min      #  $\approx 2.2e-308$  - Kleinster positiver normalisierter
sys.float_info.epsilon  #  $\approx 2.2e-16$  - Differenz zwischen 1.0 und nächstem Float
math.ulp(0.0)           #  $\approx 5e-324$  - Kleinster positiver subnormaler

# NICHT-ENDLICHE Spezialwerte
math.inf                # Positive Unendlichkeit
-math.inf               # Negative Unendlichkeit
math.nan                # Not a Number (keine Zahl)

# PRÄZISIONS-Werkzeuge
math.ulp(x)             # Unit in Last Place (Abstand zum nächsten Float)
math.nextafter(x, y)    # Nächster darstellbarer Float in Richtung y
```

Zwei Werkzeuge für präzises Grenzwert-Testen

`math.ulp(x)` - Unit in Last Place

- Gibt die **Größe des Schritts** zum nächsten darstellbaren Float von `x` zurück
- Zeigt dir "wie viel Abstand" bei dieser Größenordnung existiert

`math.nextafter(x, y)` - Nächster Float

- Gibt den **tatsächlichen nächsten Float** von `x` in Richtung `y` zurück
- Liefert den konkreten benachbarten Wert

Wichtige Beziehung:

```
math.nextafter(x, math.inf) - x == math.ulp(x)  # Bewegung Richtung  $+\infty$ 
```

`math.ulp()` und `math.nextafter()` - Beispiele

```
# Beispiel 1: Bei 1.0
x = 1.0
schrittgroesse = math.ulp(x)          # ≈ 2.22e-16
naechster_float = math.nextafter(x, 10.0) # Nächster Float Richtung 10.0

# Die Beziehung gilt:
assert naechster_float - x == schrittgroesse # ✅ Wahr!

# Beispiel 2: Abstand wächst mit Größenordnung
math.ulp(1.0)      # ≈ 2.22e-16 (winziger Abstand)
math.ulp(1e10)     # ≈ 0.001953 (viel größerer Abstand!)
math.ulp(1e100)    # ≈ 1.94e84 (riesiger Abstand!)

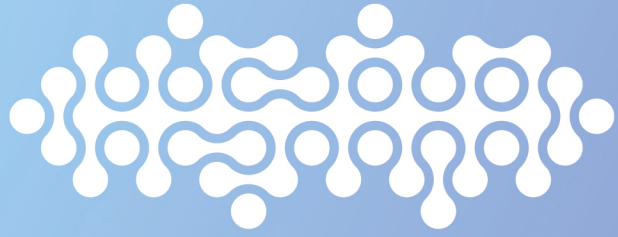
# Beispiel 3: Richtung ist wichtig für nextafter
math.nextafter(1.0, 10.0)  # 1.0000000000000002 (Richtung +∞)
math.nextafter(1.0, -10.0) # 0.9999999999999999 (Richtung -∞)
```

Anwendungsfall fürs Testen: Erstelle Werte, die **knapp** außerhalb gültiger Bereiche liegen!

! Kritischer Unterschied: `math.inf` vs `sys.float_info.max`

Aspekt	<code>math.inf</code>	<code>sys.float_info.max</code>
Typ	Nicht-endlicher Spezialwert	Größter endlicher Float
Wert	Positive Unendlichkeit	$\approx 1.8 \times 10^{308}$
Testet	Unendlichkeits-Handling	Endliche Bereichsgrenzen
Beispiel	<code>1 / math.inf == 0.0</code>	<code>sys.float_info.max * 2 == inf</code>

```
# NICHT dasselbe!  
math.inf > sys.float_info.max # True  
sys.float_info.max * 2        # Überlauf zu inf!
```



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Grenzwert-Test-Strategie

2

Grenzwert-Test-Strategie: `reciprocal(x) = 1/x`

Äquivalenzklassen → Grenzwerte

Klasse	Zu testende Grenzwerte
Positiv	<code>1e-6</code> , <code>1.0</code> , <code>1e10</code> (nahe Null, normal, groß)
Negativ	<code>-1e-6</code> , <code>-1.0</code> , <code>-1e10</code>
Null	<code>0.0</code> (sollte Exception werfen!)
Extrem	<code>sys.float_info.max</code> , <code>sys.float_info.min</code>
Spezial	<code>math.inf</code> , <code>-math.inf</code> , <code>math.nan</code>

Nicht nur die Mitte testen - teste die **Ränder**!

Beispiel: Grenzwert testen - Exakt Null

```
import pytest

def test_reciprocal_exactly_zero():
    """Grenzwert: Division durch Null sollte Exception werfen"""
    with pytest.raises(ZeroDivisionError):
        reciprocal(0.0)
```

Wichtige Punkte:

-  Teste den **exakten** Grenzwert (nicht `1e-100`)
-  Verifiziere erwartetes Verhalten (Exception, kein Crash!)
-  Verwende aussagekräftigen Testnamen

Beispiel: Grenzwert testen - Unendlichkeit

```
import math

def test_reciprocal_positive_infinity():
    """Grenzwert:  $1/\infty = 0$  (mathematischer Grenzwert)"""
    result = reciprocal(math.inf)
    assert result == 0.0

def test_reciprocal_negative_infinity():
    """Grenzwert:  $1/(-\infty) = -0$ """
    result = reciprocal(-math.inf)
    assert result == 0.0 # Python: 0.0 == -0.0 ist True
```

Testet Spezialwert-Handling, nicht Überlaufprävention!

Beispiel: Grenzwert testen - NaN (Not a Number)

```
import math

def test_reciprocal_nan():
    """Grenzwert: NaN propagiert durch Operationen"""
    result = reciprocal(math.nan)

    # ✗ FALSCH: assert result == math.nan
    # NaN == NaN ist IMMER False!

    # ✓ RICHTIG:
    assert math.isnan(result)
```

Kritisch: `nan == nan` gibt `False` zurück (IEEE 754-Standard)

Immer `math.isnan()` verwenden, um auf NaN zu prüfen!

Beispiel: Grenzwert testen - Maximaler endlicher Float

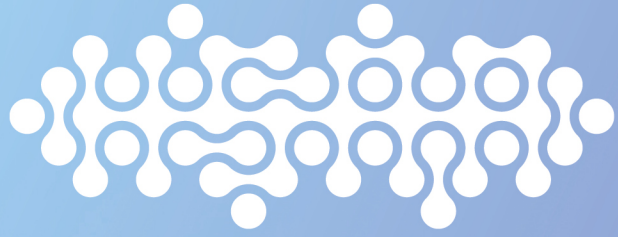
```
import sys

def test_reciprocal_max_finite_float():
    """Grenzwert: Kehrwert des größten endlichen → extrem klein

    Testet Überlaufprävention an oberer endlicher Grenze.
    Ergebnis kann zu subnormalem (denormalisiertem) Bereich unterlaufen.
    """
    result = reciprocal(sys.float_info.max)

    # Prüfe, dass Ergebnis positiv aber winzig ist
    assert result > 0
    assert result < sys.float_info.min # Unterlauf zu subnormal!
```

Ergebnis: $\approx 5.6 \times 10^{-309}$ (subnormale Zahl)



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Floats vergleichen mit `pytest.approx()`

3

Das Fließkomma-Vergleichsproblem

```
# ✗ DAS SCHLÄGT FEHL!  
def test_simple_addition():  
    result = 0.1 + 0.2  
    assert result == 0.3 # AssertionError: 0.30000000000000004 != 0.3  
  
# ✗ DAS IST FRAGIL!  
def test_with_manual_tolerance():  
    result = 0.1 + 0.2  
    assert abs(result - 0.3) < 1e-9 # Funktioniert, aber nicht empfohlen
```

Problem: Fließkomma-Arithmetik ist **nicht exakt** wegen binärer Darstellung

Lösung: Verwende `pytest.approx()` 🎯

pytest.approx() - Der richtige Weg

```
import pytest

# ✅ RICHTIG: Verwende pytest.approx()
result = 0.1 + 0.2
assert result == pytest.approx(0.3)

# ✅ Benutzerdefinierte Toleranz
assert result == pytest.approx(expected, rel=1e-9, abs=1e-12)
```

Vorteile:

- ✅ Klare Absicht
- ✅ Bessere Fehlermeldungen
- ✅ Behandelt Randfälle (NaN, Unendlichkeit)
- ✅ Funktioniert mit Arrays, Listen, Dicts!

Wie `pytest.approx()` funktioniert

Standard-Toleranzen:

- `rel=1e-6` (relativ: 0.0001%)
- `abs=1e-12` (absolut)

Vergleichsregel: Werte gleich, wenn **EINE** Toleranz erfüllt ist:

$$|\text{actual} - \text{expected}| \leq \max(\text{rel} \times |\text{expected}|, \text{abs})$$

Warum zwei Toleranzen?

- **Relativ** skaliert mit Größenordnung (gut für große Zahlen)
- **Absolut** behandelt Werte nahe Null (wo relativ versagt)

`pytest.approx()` - Beispiel mit Grenzwerten

```
import pytest
import sys

def test_reciprocal_near_zero_positive():
    """Grenzwert: Sehr klein positiv → sehr großes Ergebnis"""
    result = reciprocal(1e-6) # 0.000001
    assert result == pytest.approx(1e6, rel=1e-9) # 1000000

def test_reciprocal_very_large_positive():
    """Grenzwert: Sehr groß → sehr kleines Ergebnis"""
    result = reciprocal(1e10)
    assert result == pytest.approx(1e-10, rel=1e-9)
```

Beachten Sie: Wir spezifizieren `rel=1e-9` für engere Präzision als Standard!

Toleranzen wählen: ULP-basierte Präzision

Für hohe Präzisionsanforderungen, verwende `math.ulp()` :

```
import math
import pytest

def test_reciprocal_ulp_precision():
    """Test mit ULP (Unit in Last Place) Toleranz"""
    x = 1.0
    result = reciprocal(x)
    expected = 1.0

    # Toleranz: 10 ULPs (10 × Abstand zwischen Floats)
    tolerance = 10 * math.ulp(expected)

    assert result == pytest.approx(expected, abs=tolerance)
```

ULP: Abstand zwischen aufeinanderfolgenden darstellbaren Floats bei gegebener Größenordnung

`pytest.approx()` funktioniert mit Collections!

```
import numpy as np
import pytest

# ✅ Listen
assert [0.1 + 0.2, 0.2 + 0.4] == pytest.approx([0.3, 0.6])

# ✅ NumPy-Arrays (am häufigsten in diesem Kurs!)
result = np.array([1.0000001, 2.0000001, 3.0000001])
expected = np.array([1.0, 2.0, 3.0])
assert result == pytest.approx(expected)

# ✅ Dictionaries (vergleicht Werte)
result_dict = {"mean": 0.1 + 0.2, "std": 0.2 + 0.4}
expected_dict = {"mean": 0.3, "std": 0.6}
assert result_dict == pytest.approx(expected_dict)
```

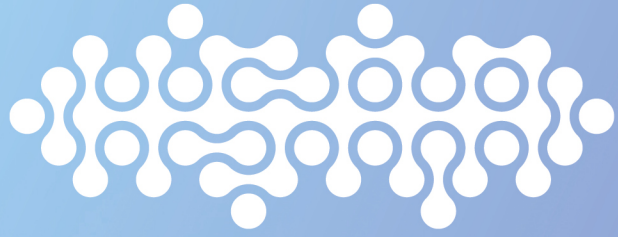
Spezialfall: Testen mit NaN

```
import math
import pytest

# ❌ FALSCH: NaN-Vergleiche schlagen immer fehl!
def test_nan_wrong():
    result = compute_invalid() # Gibt NaN zurück
    assert result == pytest.approx(math.nan) # SCHLÄGT FEHL!

# ✅ RICHTIG: Verwende math.isnan()
def test_nan_correct():
    result = compute_invalid()
    assert math.isnan(result)

# ✅ ODER: Verwende nan_ok=True (wenn Collections verglichen werden)
assert [1.0, math.nan] == pytest.approx([1.0, math.nan], nan_ok=True)
```



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Alles zusammenführen

4

Hands-On Beispiel: Vollständige Grenzwert-Testsuite

```
import sys, math, pytest

# Konstanten definieren für Lesbarkeit
FMAX = sys.float_info.max
FMIN = sys.float_info.min
INF = math.inf
NAN = math.nan

class TestReciprocalBoundaries:
    """Umfassende Grenzwert-Tests"""

    def test_exactly_zero(self):
        """Grenzwert: Division durch Null"""
        with pytest.raises(ZeroDivisionError):
            reciprocal(0.0)
```

Hands-On Beispiel: Vollständige Grenzwert-Testsuite (Forts.)

```
def test_positive_infinity(self):  
    """Grenzwert:  $1/\infty = 0$ """  
    assert reciprocal(INF) == 0.0  
  
def test_negative_infinity(self):  
    """Grenzwert:  $1/(-\infty) = -0$ """  
    assert reciprocal(-INF) == 0.0  
  
def test_nan(self):  
    """Grenzwert: NaN propagiert"""  
    assert math.isnan(reciprocal(NAN))
```

Hands-On Beispiel: Vollständige Grenzwert-Testsuite (Forts.)

```
def test_max_finite_float(self):
    """Grenzwert: Größter endlicher → winziges Ergebnis"""
    result = reciprocal(FMAX)
    assert result > 0
    assert result < FMIN # Unterlauf zu subnormal

def test_min_normalized_float(self):
    """Grenzwert: Kleinster normalisierter → riesiges Ergebnis"""
    result = reciprocal(FMIN)
    assert result == INF # Überlauf!

def test_near_zero_positive(self):
    """Grenzwert: Nahe Null → großes Ergebnis"""
    assert reciprocal(1e-6) == pytest.approx(1e6, rel=1e-9)
```

Häufige Fehler vermeiden

✗ NICHT:

- `==` für Float-Vergleiche verwenden
- Nur "Happy-Path"-Werte testen (Mitte der Bereiche)
- Spezialwerte ignorieren (`inf`, `nan`)
- Überlauf/Unterlauf vergessen

✓ TUN:

- `pytest.approx()` für alle Float-Vergleiche verwenden
- Grenzwerte systematisch testen
- Spezialwerte explizit behandeln
- Dokumentieren, warum jeder Grenzwert wichtig ist (Docstrings!)

Zusammenfassung: Ihr Test-Werkzeugkasten

Grenzwertanalyse:

1. Äquivalenzklassen identifizieren
2. **Fokus auf Grenzwerte** (nicht mittlere Werte!)
3. IEEE 754-Konstanten verwenden (`sys.float_info`, `math.inf`, `math.nan`)
4. Spezialwerte explizit testen

`pytest.approx()`:

1. **Immer** für Float-Vergleiche verwenden
2. Toleranzen bei Bedarf anpassen (`rel`, `abs`)
3. `math.ulp()` für hohe Präzisionsanforderungen verwenden
4. Funktioniert mit Arrays, Listen, Dicts!

Wichtigste Erkenntnisse

- ✓ **Bugs verstecken sich an Grenzwerten** - Teste Ränder, nicht Mitten
- ✓ **Kenne deine Float-Grenzwerte** - `sys.float_info.max`, `math.inf`, `math.nan`
- ✓ **Verwende `pytest.approx()`** - Der korrekte Weg, Floats zu vergleichen
- ✓ **Teste systematisch** - Decke alle Grenzwertfälle ab
- ✓ **Schreibe aussagekräftige Tests** - Dein zukünftiges Ich wird es dir danken!

Als Nächstes: Wende diese Techniken auf dein Road Profile Viewer-Projekt an! 🚀



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Fragen?

GROSSARTIGE TESTS FANGEN
BUGS AN DEN GRENZWERTEN

