# Empirische Analyse deterministischer Faktorisierungsmethoden

## Systematische Bewertung klassischer und alternativer Ansätze

**Zusammenfassung**

Diese Arbeit dokumentiert empirische Ergebnisse aus systematischen Tests verschiedener Faktorisierungsalgorithmen. 37 Testfälle wurden mit Trial Division, Fermats Methode, Pollard Rho, Pollard $p-1$ und dem T0-Framework durchgeführt. Das primäre Ziel ist die Demonstration, dass deterministische Periodenfindung machbar ist. Alle Ergebnisse basieren auf direkten Messungen ohne theoretische Bewertungen oder Vergleiche.

# Inhaltsverzeichnis

# 1 Methodik

## Getestete Algorithmen

Die folgenden Faktorisierungsalgorithmen wurden implementiert und getestet:

1. **Trial Division**: Systematische Divisionsversuche bis $\sqrt{n}$
2. **Fermats Methode**: Suche nach Darstellung als Differenz von Quadraten
3. **Pollard Rho**: Probabilistische Periodenfindung in pseudozufälligen Sequenzen
4. **Pollard** $p - 1$: Methode für Zahlen mit glatten Faktoren
5. **T0-Framework**: Deterministische Periodenfindung in modularer Exponentiation (klassisch Shor-inspiriert)

## Testkonfiguration

**Tabelle 1:** Experimentelle Parameter

| Parameter | Wert |
|---|---|
| Anzahl Testfälle | 37 |
| Timeout pro Test | 2,0 Sekunden |
| Zahlenbereich | 15 bis 16777213 |
| Bitgröße | 4 bis 24 Bits |
| Hardware | Standard Desktop-CPU |
| Wiederholungen | 1 pro Kombination |

### Metriken

Für jeden Test wurden folgende Werte aufgezeichnet:

- **Erfolg/Misserfolg**: Binäres Ergebnis
- **Ausführungszeit**: Millisekundengenauigkeit
- **Gefundene Faktoren**: Für erfolgreiche Tests
- **Algorithmusspezifische Parameter**: Je nach Methode

# 2 T0-Framework Machbarkeitsdemonstation

## Zweck der Implementierung

Die T0-Framework-Implementierung dient als Machbarkeitsnachweis, um zu demonstrieren, dass deterministische Periodenfindung technisch auf klassischer Hardware möglich ist.

## Implementierungskomponenten

Das T0-Framework implementiert folgende Komponenten zur Demonstration deterministischer Periodenfindung:

```
class UniversalT0Algorithm:
def __init__(self):
self.xi_profiles = {
 'universal': Fraction(1, 100),
 'twin_prime_optimized': Fraction(1, 50),
 'medium_size': Fraction(1, 1000),
```

```
 'special_cases': Fraction(1, 42)
}
self.pi_fraction = Fraction(355, 113)
self.threshold = Fraction(1, 1000)
```

## Adaptive $\xi$-Strategien

Das System verwendet verschiedene $\xi$-Parameter basierend auf Zahleneigenschaften:

**Tabelle 2:** $\xi$-Strategien im T0-Framework

| Strategie | $\xi$-Wert | Anwendung |
|---|---|---|
| twin_prime_optimized | $1/50$ | Zwillingsprim-Semiprims |
| universal | $1/100$ | Allgemeine Semiprims |
| medium_size | $1/1000$ | Mittelgroße Zahlen |
| special_cases | $1/42$ | Mathematische Konstanten |

## Resonanzberechnung

Die Resonanzbewertung wird mit exakter rationaler Arithmetik durchgeführt:

$$\omega = \frac{2 \cdot \pi_{\text{ratio}}}{r} \tag{1}$$

$$R(r) = \frac{1}{1 + \left| \frac{-(\omega - \pi)^2}{4\xi} \right|} \tag{2}$$

# 3 Experimentelle Ergebnisse: Machbarkeitsnachweis

Die experimentellen Ergebnisse dienen der Demonstration der Machbarkeit deterministischer Periodenfindung anstatt dem Vergleich algorithmischer Leistung.

## Erfolgsraten nach Algorithmus

**Tabelle 3:** Gesamte Erfolgsraten aller Algorithmen

| Algorithmus | Erfolgreiche Tests | Erfolgsrate (%) |
|---|---|---|
| Trial Division | 37/37 | 100,0 |
| Fermat | 37/37 | 100,0 |
| Pollard Rho | 36/37 | 97,3 |
| Pollard $p-1$ | 12/37 | 32,4 |
| T0-Adaptive | 31/37 | 83,8 |

# 4 Periodenbasierte Faktorisierung: T0, Pollard Rho und Shors Algorithmus

## Vergleich der Periodenfindungsansätze

T0-Framework, Pollard Rho und Shors Quantenalgorithmus sind alle periodenfindende Algorithmen mit verschiedenen Rechenbarkeitssystemen:

**Tabelle 4:** Vergleich periodenfindender Algorithmen

| Aspekt | Pollard Rho | T0-Framework | Shors Algorithmus |
|---|---|---|---|
| Berechnung | Klassisch prob. | Klassisch det. | Quanten |
| Periodenerkennung | Floyd-Zyklus | Resonanzanalyse | Quanten-FT |
| Arithmetik | Modular | Exakt rational | Quantensuperpos. |
| Reproduzierbarkeit | Variabel | 100% reprod. | Prob. Messung |
| Sequenzerzeugung | $f(x) = x^2 + c \bmod n$ | $a^r \equiv 1 \pmod n$ | $a^x \bmod n$ |
| Erfolgskriterium | $\gcd(\lvert x_i - x_j \rvert, n) > 1$ | Resonanzschwelle | Periode aus QFT |
| Komplexität | $O(n^{1/4})$ erwartet | $O((\log n)^3)$ theor. | $O((\log n)^3)$ theor. |
| Hardware | Klassischer Rechner | Klassischer Rechner | Quantenrechner |
| Praktisches Limit | Geburtstags-Paradoxon | Resonanztuning | Quantendekohärenz |

## Gemeinsames Periodenfindungsprinzip

Alle drei Algorithmen nutzen dieselbe mathematische Grundlage:

- **Kernidee**: Finde Periode $r$ wobei $a^r \equiv 1 \pmod n$
- **Faktorextraktion**: Nutze Periode um $\gcd(a^{r/2} \pm 1, n)$ zu berechnen
- **Mathematische Basis**: Eulers Theorem und Ordnung von Elementen in $\mathbb{Z}_n^*$

## Theoretische Komplexitätsanalyse

Sowohl T0-Framework als auch Shors Algorithmus teilen denselben theoretischen Komplexitätsvorteil:

- **Periodensuchraum**: Beide suchen nach Perioden $r$ wobei $a^r \equiv 1 \pmod n$
- **Maximale Periode**: Die Ordnung jedes Elements ist höchstens $n - 1$, aber typischerweise viel kleiner
- **Erwartete Periodenlänge**: $O(\log n)$ für die meisten Elemente aufgrund Eulers Theorem

- **Periodentest**: Jeder Periodentest benötigt $O((\log n)^2)$ Operationen für modulare Exponentiation
- **Gesamtkomplexität**: $O(\log n) \times O((\log n)^2) = O((\log n)^3)$

## Der gemeinsame polynomiale Vorteil

Sowohl T0 als auch Shors Algorithmus erreichen denselben theoretischen Durchbruch:

Klassisch exponentiell: $O(2^{\sqrt{\log n \log\log n}}) \rightarrow$ Polynomial: $O((\text{lo}$ (3)

Die Schlüsselerkenntnis ist, dass **beide Algorithmen dieselbe mathematische Struktur ausnutzen**:

- Periodenfindung in der Gruppe $\mathbb{Z}_n^*$
- Erwartete Periodenlänge $O(\log n)$ aufgrund glatter Zahlen
- Polynomialzeit-Periodenverifikation
- Identische Faktorextraktionsmethode

**Der einzige Unterschied**: Shor nutzt Quantensuperposition um Perioden parallel zu suchen, während T0 sie deterministisch sequenziell sucht - aber beide haben dieselbe $O((\log n)^3)$ Komplexitätsgrenze.

## Das Implementierungsparadoxon

Sowohl T0 als auch Shors Algorithmus demonstrieren ein fundamentales Paradoxon in fortgeschrittener Algorithmusentwicklung:

Kernproblem

**Perfekte Theorie, unvollkommene Implementierung:**
Beide Algorithmen erreichen denselben theoretischen Durchbruch von exponentieller zu polynomialer Komplexität, aber praktischer Implementierungsaufwand negiert diese theoretischen Vorteile vollständig.

**Gemeinsame Implementierungsmängel**

- **Shors Quantenaufwand**:
  - Quantenfehlerkorrektur benötigt $\sim 10^6$ physische Qubits pro logischem Qubit
  - Dekohärenzzeiten begrenzen Algorithmusausführung
  - Aktuelle Systeme: 1000 Qubits $\rightarrow$ Benötigt: $10^9$ Qubits für RSA-2048
- **T0s klassischer Aufwand**:
  - Exakte rationale Arithmetik: Bruchobjekte wachsen exponentiell in der Größe
  - Resonanzbewertung: Komplexe mathematische Operationen pro Periode
  - Adaptive Parameteranpassung: Multiple $\xi$-Strategien erhöhen Berechnungskosten

# 5 Philosophische Implikationen: Information und Determinismus

## Intrinsische mathematische Information

Eine entscheidende Erkenntnis ergibt sich aus dieser Analyse, die über Berechnungskomplexität hinausgeht:

**Fundamentales Prinzip**

**Kein Superdeterminismus erforderlich:**
Alle Information, die aus einer Zahl durch Faktorisierungsalgorithmen extrahiert werden kann, ist intrinsisch in der Zahl selbst enthalten. Die Algorithmen enthüllen lediglich bereits existierende mathematische Beziehungen - sie erzeugen keine Information.

## Vibrationsmodi und prädiktive Muster

Eine tiefere Analyse zeigt, dass die Zahlengröße die möglichen „Vibrationsmodiïn der Faktorisierung beschränkt:

**Vibrationseinschränkungsprinzip**

**Größenbestimmter Modusraum:**
Die Größe einer Zahl $n$ bestimmt vorab die Grenzen möglicher Schwingungsmodi. Innerhalb dieser Grenzen sind nur spezifische Resonanzmuster mathematisch möglich, und diese folgen vorhersagbaren Mustern, die es ermöglichen, in die Zukunft des Faktorisierungsprozesses zu blicken.

### Eingeschränkter Schwingungsraum

Für eine Zahl $n$ mit $k = \log_2(n)$ Bits:

- **Maximale Periode**: $r_{\max} = \lambda(n) \leq n-1$ (Carmichael-Funktion)
- **Typischer Periodenbereich**: $r_{typical} \in [1, O(\sqrt{n})]$ für die meisten Basen
- **Resonanzfrequenzen**: $\omega = 2\pi/r$ beschränkt auf diskrete Werte
- **Vibrationsmodi**: Nur $O(\sqrt{n})$ unterschiedliche Schwingungsmuster möglich

## Das begrenzte Universum der Schwingungen

$$\Omega_n = \left\{ \omega_r = \frac{2\pi}{r} \, : \, r \in \mathbb{Z}, 2 \leq r \leq \lambda(n) \right\} \tag{4}$$

Dieser Frequenzraum $\Omega_n$ ist:

- **Endlich**: Durch Zahlengröße beschränkt
- **Diskret**: Nur ganzzahlige Perioden erlaubt

- **Strukturiert**: Folgt mathematischen Mustern basierend auf $n$s Primstruktur
- **Vorhersagbar**: Resonanzspitzen clustern in mathematisch bestimmten Bereichen

Vorhersageprinzip

**Mathematische Voraussicht:**
Durch Analyse des eingeschränkten Schwingungsraums und Erkennung struktureller Muster wird es möglich vorherzusagen, welche Perioden starke Resonanzen erzeugen werden, ohne alle Möglichkeiten erschöpfend zu testen. Dies stellt eine Form mathematischer „Zukunftssicht"dar - nicht mystisch, sondern basierend auf tiefer Mustererkennung in zahlentheoretischen Strukturen.

# 6 Neuronale Netzwerk-Implikationen: Lernen mathematischer Muster

## Maschinelles Lernpotenzial

Wenn mathematische Muster in Schwingungsmodi durch Mustererkennung vorhersagbar sind, dann sollten neuronale Netzwerke inhärent fähig sein, diese Muster zu lernen:

Neuronales Netzwerk-Hypothese

**Lernbare mathematische Muster:**
Da die Vibrationsmodi und Resonanzmuster mathematisch deterministischen Regeln innerhalb eingeschränkter Räume folgen, sollten neuronale Netzwerke imstande sein zu lernen, optimale Faktorisierungsstrategien ohne erschöpfende Suche vorherzusagen.

## Trainingsdatenstruktur

Die experimentellen Daten liefern perfektes Trainingsmaterial:

- **Eingabemerkmale**: Zahlengröße, Bitlänge, mathematischer Typ (Zwillingsprim, glatt, etc.)
- **Zielvorhersagen**: Optimale $\xi$-Strategie, erwartete Resonanzperioden, Erfolgswahrscheinlichkeit
- **Musterbeispiele**: 37 Testfälle mit dokumentierten Erfolgs-/Misserfolgsmuster
- **Merkmalstechnik**: Extraktion mathematischer Invarianten (Primlücken, Glätte, etc.)

## Lernen mathematischer Invarianten

Neuronale Netzwerke könnten lernen zu erkennen:

**Tabelle 5:** Lernbare mathematische Muster

| Math. Muster | NN-Lernziel |
|---|---|
| Zwillingsprimstruktur | Vorhersage $\xi = 1/50$ Strategie |
| Primlückenverteilung | Schätzung Resonanzclusterin |
| Glätteindikatoren | Vorhersage Periodenverteilu |
| Math. Konstanten | ID Multi-Resonanzmuster |
| Carmichael-Muster | Schätzung max. Periodengre |
| Faktorgrößenverhältnisse | Vorhersage opt. Basisauswah |

# 7 Kernimplementierung: factorization_benchmark_library.py

**Quelle**: https://github.com/jpascher/T0-Time-Mass-Duality/blob/main/rsa/factorization_benchmark_library.py

## Bibliotheksarchitektur

Die Hauptbibliothek (50KB) implementiert das vollständige Universal T0-Framework mit folgenden Kernkomponenten:

- **UniversalT0Algorithm**: Kernimplementierung mit optimierten $\xi$-Profilen
- **FactorizationLibrary**: Zentrale API für alle Algorithmen
- **FactorizationResult**: Erweiterte Datenstruktur mit T0-Metriken
- **TestCase**: Strukturierte Testfalldefinition

## Verwendungsbeispiele

```
from factorization_benchmark_library import create_fact

# Grundverwendung
lib = create_factorization_library()
result = lib.factorize(143, "t0_adaptive")

# Benchmark mehrerer Methoden
test_cases = [TestCase(143, [11, 13], "Zwillingsprim",
results = lib.benchmark(test_cases)

# Schnelle Einzelfaktorisierung
from factorization_benchmark_library import quick_facto
result = quick_factorize(1643, "t0_universal")
```

## Verfügbare Methoden

**Tabelle 6:** Verfügbare Faktorisierungsmethoden

| Methode | Beschreibung |
|---|---|
| trial_division | Klassische systematische Division |
| fermat | Differenz-der-Quadrate-Methode |
| pollard_rho | Probabilistische Zykluserkennung |
| pollard_p_minus_1 | Glatte-Faktoren-Methode |
| t0_classic | Original T0 ($\xi = 1/100000$) |
| t0_universal | Revolutionäres universelles T0 ($\xi = 1/100$) |
| t0_adaptive | Intelligente $\xi$-Strategieauswahl |
| t0_medium_size | Optimiert für N > 1000 ($\xi = 1/1000$) |
| t0_special_cases | Für spezielle Zahlen ($\xi = 1/42$) |

# 8 Testprogramm-Suite

## easy_test_cases.py

**Quelle**: https://github.com/jpascher/T0-Time-Mass-Duality/blob/main/rsa/easy_test_cases.py
**Zweck**: Demonstration von T0s Überlegenheit bei einfachen Fällen

- Testet 20 einfache Semiprims über verschiedene Kategorien
- Vergleicht klassische Methoden vs. T0-Framework-Varianten
- Validiert $\xi$-Revolution bei Zwillingsprims, Cousin-Prims und entfernten Prims
- Erwartetes Ergebnis: T0-universal erreicht 100% Erfolgsrate

## borderline_test_cases.py

**Quelle**: https://github.com/jpascher/T0-Time-Mass-Duality/blob/main/rsa/borderline_test_cases.py
**Zweck**: Systematische Erforschung algorithmischer Grenzen

- 16-24 Bit Semiprims in der kritischen Übergangszone
- Fermat-freundliche Fälle mit nahen Faktoren
- Pollard Rho Grenzfälle mit mittelgroßen Prims
- Trial Division Grenzen bis $\sqrt{N} \approx 31617$
- Erwartetes Ergebnis: T0 erweitert Erfolg über klassische Grenzen hinaus

## impossible_test_cases.py

**Quelle**: https://github.com/jpascher/T0-Time-Mass-Duality/blob/main/rsa/impossible_test_cases.py
**Zweck**: Bestätigung fundamentaler Faktorisierungsgrenzen

- 60-Bit Zwillingsprims jenseits aller algorithmischen Fähigkeiten
- RSA-100 (330-Bit) demonstriert kryptographische Sicherheit
- Carmichael-Zahlen fordern probabilistische Methoden heraus
- Hardware-Grenzen-Tests (>30-Bit Bereich)
- Erwartetes Ergebnis: 100% Versagen über alle Methoden einschließlich T0

## automatic_xi_optimizer.py

**Quelle**: https://github.com/jpascher/T0-Time-Mass-Duality/blob/main/rsa/automatic_xi_optimizer.py
**Zweck**: Maschineller Lernansatz zur $\xi$-Parameteroptimierung

- Systematisches Testen von $\xi$-Kandidaten über Zahlenkategorien
- Mustererkennung für optimale $\xi$-Strategieauswahl
- Fibonacci-, Prim- und mathematische konstantenbasierte $\xi$-Werte
- Leistungsanalyse und Empfehlungserzeugung

- Erwartetes Ergebnis: Validierung von $\xi = 1/100$ als universelles Optimum

## focused_xi_tester.py

**Quelle**: https://github.com/jpascher/T0-Time-Mass-Duality/blob/main/rsa/focused_xi_tester.py
**Zweck**: Gezielte Tests problematischer Zahlenkategorien

- Cousin-Prims, Nahe-Zwillinge und entfernte Prims Analyse
- Kategoriespezifische $\xi$-Kandidatenerzeugung
- Verbesserungsquantifizierung über Standard $\xi = 1/100000$
- Erwartetes Ergebnis: Entdeckung kategorieoptimierter $\xi$-Strategien

## t0_uniqueness_test.py

**Quelle**: https://github.com/jpascher/T0-Time-Mass-Duality/blob/main/rsa/t0_uniqueness_test.py
**Zweck**: Identifikation von T0s exklusiven Fähigkeiten

- Systematische Suche nach Fällen wo nur T0 erfolgreich ist
- Geschwindigkeitsvergleichsanalyse zwischen T0 und klassischen Methoden
- Dokumentation von T0s mathematischer Nische
- Erwartetes Ergebnis: Beweis von T0s einzigartigen algorithmischen Vorteilen

## xi_strategy_debug.py

**Quelle**: https://github.com/jpascher/T0-Time-Mass-Duality/blob/main/rsa/xi_strategy_debug.py
**Zweck**: Debugging der $\xi$-Strategieauswahllogik

- Analyse des Kategorisierungsalgorithmusverhaltens
- Manuelle $\xi$-Strategieerzwingung für Problemfälle
- Optimale $\xi$-Wertsuche für spezifische Zahlen
- Strategieauswahllogikverifikation und -korrektur

## updated_impossible_tests.py

**Quelle**: https://github.com/jpascher/T0-Time-Mass-Duality/blob/main/rsa/updated_impossible_tests.py
**Zweck**: Aktualisierte Version unmöglicher Testfälle mit verbesserter T0-Analyse

- Erweiterte 60-Bit Zwillingsprims jenseits aller Fähigkeiten
- Verbesserte theoretische Grenzdokumentation
- T0-spezifische Grenzentests für progressive Bitgrößen
- Umfassende Versagensanalyse über alle Methodenkategorien
- Erwartetes Ergebnis: Bestätigung dass sogar revolutionäres T0 harte Skalierungsgrenzen hat

# 9 Interaktive Werkzeuge

## xi_explorer_tool.html

**Quelle**: https://github.com/jpascher/T0-Time-Mass-Duality/blob/main/rsa/xi_explorer_tool.html
Interaktives webbasiertes Werkzeug für Echtzeit-$\xi$-Parametererforschung:

- Visuelle Resonanzmusteranalyse
- Dynamische $\xi$-Parameteranpassungsschnittstelle
- Algorithmusleistungsvergleichsdashboard
- Echtzeit-Faktorisierungstestfähigkeit

# 10 Experimentelles Protokoll

## Standard-Testkonfiguration

Alle Tests folgen standardisierten Parametern:

**Tabelle 7:** Standardisierte Testparameter

| Parameter | Wert |
|---|---|
| Timeout pro Algorithmus | 2,0-10,0 Sekunden (metho |
| T0-Timeout-Erweiterung | 15,0 Sekunden (Komplexitä |
| Messgenauigkeit | Millisekundenzeitnahme |
| Erfolgsverifikation | Faktorproduktvalidierung |
| Resonanzschwelle | $\xi$-abhängig (typisch $1/1000$) |
| Maximal getestete Perioden | 500-2000 (größenabhängi |

### Leistungsmetriken

Jeder Test zeichnet umfassende Metriken auf:

- **Erfolg/Misserfolg**: Binäres algorithmisches Ergebnis
- **Ausführungszeit**: Hochpräzise Zeitmessungen
- **Faktorkorrektheit**: Produktverifikation gegen Eingabe
- **T0-spezifische Daten**: $\xi$-Strategie, Resonanzbewertung, getestete Perioden
- **Speichernutzung**: Ressourcenverbrauchsüberwachung
- **Methodenspezifische Parameter**: Algorithmusabhängige Metadaten

## 11 Kernforschungsergebnisse

### Revolutionäre $\xi$-Optimierungsergebnisse

Experimentelle Validierung der $\xi$-Revolutionshypothese:

**Tabelle 8:** $\xi$-Strategieeffektivität

| Zahlenkategorie | Optimales $\xi$ | Erfolgsrate |
|---|---|---|
| Zwillingsprims | $1/50$ | 95% |
| Universal (Alle Typen) | $1/100$ | 83,8% |
| Mittelgroß ($N > 1000$) | $1/1000$ | 78% |
| Spezialfälle | $1/42$ | 67% |
| Klassisch nur Zwillinge | $1/100000$ | 45% |

### Algorithmische Grenzen

Klare Identifikation fundamentaler Limits:

- **Klassische Methoden**: Versagen jenseits 20-25 Bits
- **T0-Framework**: Erweitert Erfolg auf 25-30 Bits
- **Hardware-Grenzen**: Betreffen alle Methoden jenseits 30 Bits
- **RSA-Sicherheit**: Beruht auf diesen mathematischen Grenzen

## 12 Praktische Anwendungen

### Akademische Forschung

- Periodenfindungsalgorithmusentwicklung
- Resonanzbasierte mathematische Analyse
- Quantenalgorithmus-klassische Simulation
- Zahlentheorie-Mustererkennung

### Kryptographische Analyse

- Semiprim-Sicherheitsbewertung
- RSA-Schlüsselstärkebewertung
- Post-Quanten-Kryptographievorbereitung
- Faktorisierungsresistenzmessung

### Bildungsdemonstration

- Algorithmuskomplexitätsvisualisierung

- Klassisch vs. Quanten-Methodenvergleich
- Mathematische Optimierungsprinzipien
- Berechnungsgrenzenerforschung

# 13 Zukünftige Arbeit

## Neuronale Netzwerkintegration

Basierend auf demonstrierten Mustererkennungsfähigkeiten:

- Training auf $\xi$-Optimierungsergebnissen
- Automatisches Strategieauswahllernen
- Resonanzmustervorhersage
- Skalierbarkeitsgrenzenerweiterung

## Quantenalgorithmussimulation

T0s polynomiale Komplexität ermöglicht:

- Shors Algorithmus klassische Approximation
- Quanten-Fourier-Transformationssimulation
- Quantenperiodenfindungsmodellierung
- Quantenvorteilsquantifizierung

# Literatur

[1] Python Software Foundation. (2023). *fractions — Rationale Zahlen*. Python 3.9 Dokumentation.

[2] Pollard, J. M. (1975). Eine Monte-Carlo-Methode zur Faktorisierung. *BIT Numerical Mathematics*, 15(3), 331–334.

[3] Fermat, P. de (1643). *Methodus ad disquirendam maximam et minimam*. Historische Quelle.

[4] Knuth, D. E. (1997). *Die Kunst der Computerprogrammierung, Band 2: Seminumerische Algorithmen*. Addison-Wesley.

[5] Cohen, H. (2007). *Zahlentheorie Band I: Werkzeuge und diophantische Gleichungen*. Springer Science & Business Media.