Geometrische Steuerung des Quantenvakuums

Anisotropie und Verschränkung in modifizierten Randbedingungen

Wissenschaftliche Abhandlung

Klaus H. Dieckmann

Oktober 2025

Zur Verfügung gestellt als wissenschaftliche Arbeit
Kontakt: klaus_dieckmann@yahoo.de

Metadaten zur wissenschaftlichen Arbeit

Titel:	Geometrische Steuerung des Quantenvakuums
Untertitel:	Anisotropie und Verschränkung in modifizierten Randbedingungen
Autor:	Klaus H. Dieckmann
Kontakt:	klaus_dieckmann@yahoo.de
Phone:	0176 50 333 206
ORCID:	0009-0002-6090-3757
DOI:	10.5281/zenodo.17256837
Version:	Oktober 2025
Lizenz:	CC BY-NC-ND 4.0
Zitatweise:	Dieckmann, K.H. (2025). Geometrische Steuerung des Quantenvakuums

Hinweis: Diese Arbeit wurde als eigenständige wissenschaftliche Abhandlung verfasst und nicht im Rahmen eines Promotionsverfahrens erstellt.

Abstract

Diese Arbeit untersucht die geometrische Steuerung des Quantenvakuums in einem anisotropen Kavitätsmodell, parametrisiert durch den Deformationsparameter γ in der Randbedingung $x^2 + y^2 + \gamma(x^4 + y^4) = 1$. Die Analyse zeigt, dass γ als universeller Ordnungsparameter fungiert und mehrere fundamentale Aspekte vereint:

1. Die **Casimir-Energie** folgt $E_C(\gamma) \approx E_0 - C\gamma^2$ und reproduziert das experimentell beobachtete Casimir-Drehmoment
2. Ein radialer tanh-Übergang erzeugt eine **geometrisch induzierte Domain Wall** mit lokalisierten Vakuumzuständen
3. Diese Zustände werden als **geometrisch lokalisierte Qubits** identifiziert und bieten eine robuste Plattform für Quanteninformationsverarbeitung
4. Die **Verschränkungsentropie** zeigt quadratische Abhängigkeit $\Delta S_{\text{ent}}(\gamma) \propto -\gamma^2$ und ermöglicht geometrische Kontrolle der Quanteninformation

Das Modell etabliert einen vereinheitlichten Rahmen, der Energie, Verschränkung und Quantencomputing verbindet. Die lokalisierten Vakuumzustände in der Domain Wall erweisen sich als vielversprechende Bausteine für fehlertolerante Quantenprozessoren in photonischen und supraleitenden Systemen. Die Arbeit liefert damit experimentell zugängliche Konzepte für Quantensimulation und Metamaterialdesign auf Basis etablierter physikalischer Prinzipien.

Inhaltsverzeichnis

Teil I

Empirische Grundlage

Kapitel 1

Einleitung

Das Quantenvakuum ist kein passiver Hintergrund, sondern ein dynamisches Medium, dessen Eigenschaften durch geometrische Randbedingungen geformt werden können. Diese Erkenntnis, die auf die theoretische Arbeit von Hendrik Casimir (1948) zurückgeht, wurde in den letzten Jahrzehnten durch präzise Experimente eindrucksvoll bestätigt. Der Casimir-Effekt demonstriert, dass die Nullpunktsenergie des elektromagnetischen Feldes messbare Kräfte und Drehmomente hervorruft, sobald die Symmetrie des Raumes durch materielle Grenzen gebrochen wird.

Während die klassische Formulierung des Effekts parallele, isotrope Platten betrachtet, zeigt die moderne Forschung, dass die reiche Struktur des Quantenvakuums erst in vollem Umfang sichtbar wird, wenn man anisotrope oder nichttriviale Randbedingungen einführt. Insbesondere das kürzlich gemessene Casimir-Drehmoment belegt, dass die Vakuumenergie empfindlich auf die relative Ausrichtung anisotroper Materialien reagiert. Diese Beobachtung legt nahe, dass die Geometrie selbst als Steuerparameter für das Quantenvakuum fungieren kann.

Vor diesem Hintergrund stellt sich die zentrale Frage dieser Arbeit: **Inwieweit lässt sich die Struktur des Quantenvakuums systematisch durch gezielte geometrische Deformationen kontrollieren, und welche universellen physikalischen Phänomene, von geometrischen Defekten über Quantenverschränkung bis hin zu Anwendungen in der Quanteninformationsverarbeitung, treten dabei hervor?**

Um diese Frage zu beantworten, führen wir ein parametrisches geometrisches Modell ein, das die Abweichung von der Rotationssymmetrie durch einen einzigen, kontinuierlichen Parameter γ beschreibt. Die zugrundeliegende Kavität

wird definiert durch die Gleichung

$$x^2 + y^2 + \gamma(x^4 + y^4) = 1,$$

wobei $\gamma = 0$ den isotropen Kreis und $\gamma > 0$ eine zunehmend anisotrope Form repräsentiert.

Obwohl das Modell in zwei Raumdimensionen formuliert ist, ist es als effektive Beschreibung für Systeme gedacht, in denen die Dynamik in einer Richtung unterdrückt ist, etwa in dünnen Metamaterialschichten, integrierten photonischen Plattformen oder planaren supraleitenden Schaltkreisen. In solchen Systemen bestimmt die laterale Geometrie die relevanten Casimir-Effekte, und die Reduktion auf $2 + 1$ Dimensionen ist physikalisch gut motiviert.

Die zentrale Hypothese dieser Arbeit lautet: **Der Geometrieparameter γ fungiert als universeller Ordnungsparameter, der nicht nur die Casimir-Energie, sondern auch geometrische Eigenschaften des Vakuums, seine Quantenverschränkungsstruktur und seine Eignung als Plattform für Quantencomputer bestimmt.**

Die vorliegende Arbeit gliedert sich in **vier thematische Schwerpunkte**:

1. **Empirische Grundlage**: Wir etablieren das γ-Modell durch Berechnung der Casimir-Energie und validieren es anhand experimenteller Daten.
2. **Geometrische Vertiefung**: Der radiale Übergang im Profil $\gamma(r)$ wird als geometrisch induzierte Domain Wall interpretiert, die lokalisierte Vakuumzustände hervorbringt.
3. **Quanteninformationsanwendung**: Wir zeigen, dass diese lokalisierten Zustände als **geometrische Qubits** dienen können und analysieren ihre Kohärenzeigenschaften sowie experimentelle Realisierbarkeit.
4. **Fundamentale Perspektive**: Schließlich untersuchen wir, wie die geometrische Deformation die Verschränkungsentropie moduliert und diskutieren die Verbindung zu informationstheoretischen Prinzipien.

Ziel dieser Arbeit ist es zu zeigen, dass eine scheinbar einfache geometrische Deformation weitreichende Konsequenzen für die Struktur des Quantenvakuums hat, von messbaren Kräften über geometrische Zustände bis hin zu technologisch nutzbaren Qubits. Damit leistet sie einen Beitrag zu einem Paradigmenwechsel: **vom Quantenvakuum als passivem Reservoir hin zu einem aktiv gestaltbaren Medium.**

Kapitel 2

Grundlagen: Quantenfeldtheorie im gekrümmten Raum und mit Randbedingungen

Die theoretische Beschreibung des Quantenvakuums unter modifizierten Randbedingungen erfordert eine Synthese aus relativistischer Quantenfeldtheorie, Spektraltheorie partieller Differentialgleichungen und semi-klassischer Gravitation. Dieses Kapitel stellt die notwendigen Konzepte bereit, um in den folgenden Kapiteln das Zusammenspiel zwischen geometrischer Deformation und Vakuumstruktur systematisch zu analysieren. Der Fokus liegt dabei auf flacher Raumzeit; gekrümmte Hintergründe werden nur insoweit behandelt, als sie für das Verständnis von Energiebedingungen relevant sind.

2.1 Das Quantenvakuum in Minkowski-Raumzeit

In der Quantenfeldtheorie (QFT) ist das Vakuum nicht als leerer Raum, sondern als der Grundzustand eines Quantenfeldes definiert. Für ein freies, masseloses Skalarfeld $\phi(x)$ in $(3+1)$-dimensionaler Minkowski-Raumzeit lautet die Feldentwicklung

$$\phi(t, \mathbf{x}) = \int \frac{d^3k}{(2\pi)^3} \frac{1}{\sqrt{2\omega_{\mathbf{k}}}} \left(a_{\mathbf{k}} e^{-i\omega_{\mathbf{k}} t + i\mathbf{k}\cdot\mathbf{x}} + a_{\mathbf{k}}^\dagger e^{i\omega_{\mathbf{k}} t - i\mathbf{k}\cdot\mathbf{x}} \right), \tag{2.1}$$

mit $\omega_{\mathbf{k}} = |\mathbf{k}|$ und den üblichen Kommutatorrelationen $[a_{\mathbf{k}}, a_{\mathbf{k}'}^\dagger] = (2\pi)^3 \delta^{(3)}(\mathbf{k} - \mathbf{k}')$. Der Vakuumzustand $|0\rangle$ ist durch $a_{\mathbf{k}} |0\rangle = 0$ für alle $\mathbf{k}$ definiert.

Die Vakuumenergiedichte ergibt sich formal aus dem Erwartungswert des Ha-

miltonoperators:

$$\langle 0|T_{00}|0\rangle = \frac{1}{2}\int \frac{d^3k}{(2\pi)^3}\,\omega_{\mathbf{k}}. \tag{2.2}$$

Diese Größe divergiert ultraviolett und wird in der freien Theorie durch Renormierung eliminiert. In Anwesenheit von Randbedingungen oder Hintergrundfeldern ändert sich jedoch das Spektrum der erlaubten Moden, und die *Differenz* zur freien Vakuumenergie wird endlich und physikalisch messbar. Dies ist der Ursprung des Casimir-Effekts.

2.2 Randbedingungen und Modenspektren

Randbedingungen modifizieren das Spektrum des Laplace-Operators $-\nabla^2$ auf einem beschränkten Gebiet $\Omega \subset \mathbb{R}^d$. Für Dirichlet-Randbedingungen ($\phi|_{\partial\Omega} = 0$) ist das Eigenwertproblem

$$-\nabla^2\psi_n(\mathbf{x}) = k_n^2\psi_n(\mathbf{x}), \quad \psi_n|_{\partial\Omega} = 0, \tag{2.3}$$

wohlgestellt, und das Spektrum $\{k_n^2\}$ ist diskret, positiv und wächst asymptotisch wie $k_n \sim n^{1/d}$ (Weyl-Gesetz).

Die Casimir-Energie für ein masseloses Skalarfeld in $d+1$ Dimensionen ist dann definiert als

$$E_C = \frac{\hbar c}{2}\sum_{n=1}^{\infty} k_n, \tag{2.4}$$

wobei die Summe im Allgemeinen divergiert und einer Regularisierung bedarf (z. B. Zeta-Funktion- oder Cut-off-Regularisierung). Die physikalisch relevante Größe ist stets die *Differenz* zur Energie im unbeschränkten Raum oder zwischen zwei Konfigurationen.

Für rotationssymmetrische Gebiete (z. B. Kreis, Kugel) ist das Spektrum analytisch zugänglich. Für anisotrope Geometrien, wie das in dieser Arbeit untersuchte γ-deformierte Gebiet, muss das Eigenwertproblem numerisch gelöst werden, z. B. mittels Finite-Elemente-Methoden.

2.3 Der Casimir-Effekt: Theorie und experimentelle Bestätigung

Der von H. B. G. Casimir 1948 vorhergesagte Effekt beschreibt eine attraktive Kraft zwischen zwei unendlich ausgedehnten, perfekt leitenden Platten im Vakuum, verursacht durch die Modifikation des elektromagnetischen Nullpunkt-

spektrums [6]. Für den parallelen Plattenabstand a lautet die Energiedichte

$$\rho_C = -\frac{\hbar c \pi^2}{720 a^4}. \tag{2.5}$$

Diese negative Energiedichte verletzt die klassische schwache Energiebedingung lokal, ist jedoch im Rahmen der semi-klassischen Gravitation zulässig.

Experimentell wurde der Effekt erstmals 1997 von Lamoreaux mit hoher Präzision bestätigt [10]. In den letzten zwei Jahrzehnten wurden zahlreiche Varianten untersucht, darunter Kugel-Platte-Geometrien, realistische Materialien (mit endlicher Leitfähigkeit und Temperaturkorrekturen) sowie anisotrope Systeme. Ein Meilenstein war 2018 die direkte Messung des *Casimir-Drehmoments* zwischen doppelbrechenden Materialien durch Munday et al. [14], das belegt, dass die Vakuumenergie empfindlich auf die relative Orientierung anisotroper Strukturen reagiert.

2.4 Quantenungleichungen und die Lokalität negativer Energiedichte

Obwohl negative Energiedichten im Casimir-Effekt auftreten, sind sie strengen quantenfeldtheoretischen Beschränkungen unterworfen. Ford und Roman zeigten in den 1990er Jahren, dass für jede zeitartige Geodäte mit Tangentialvektor u^μ gilt [7]:

$$\int_{-\infty}^{\infty} d\tau \, \langle T_{\mu\nu} u^\mu u^\nu \rangle f(\tau) \geq -\frac{C}{\tau_0^4}, \tag{2.6}$$

wobei $f(\tau)$ ein positives Testfunktionsfenster der Breite τ_0 ist und C eine universelle Konstante. Diese sogenannten *Quantenungleichungen* implizieren, dass negative Energiedichte stets durch positive Anteile kompensiert werden muss und ihre räumliche Ausdehnung invers zur Stärke skaliert.

Für makroskopische Wurmloch-Modelle stellt dies eine fundamentale Hürde dar. Im Kontext dieser Arbeit bedeutet es jedoch, dass unser geometrisches Modell nur dann physikalisch sinnvoll ist, wenn die durch γ induzierte negative Energiedichte lokal begrenzt bleibt, eine Eigenschaft, die durch den glatten tanh-Übergang in Kapitel 4 sichergestellt wird.

2.5 Geometrische Formulierung: Laplace-Operator auf anisotropen Gebieten

Um die Abhängigkeit der Casimir-Energie von der Geometrie zu quantifizieren, betrachten wir ein zweidimensionales Gebiet $\Omega_\gamma \subset \mathbb{R}^2$, definiert durch

$$x^2 + y^2 + \gamma(x^4 + y^4) = 1, \quad \gamma \geq 0. \tag{2.7}$$

Für $\gamma = 0$ reduziert sich Ω_0 auf die Einheitskreisscheibe; für $\gamma > 0$ entsteht eine anisotrope, vierfach symmetrische Form.

Der Laplace-Operator $-\nabla^2$ auf Ω_γ mit Dirichlet-Randbedingungen besitzt ein diskretes Spektrum $\{k_n^2(\gamma)\}$. Die Casimir-Energie (in $2+1$ Dimensionen) ist dann

$$E_C(\gamma) = \frac{\hbar c}{2} \sum_{n=1}^{\infty} k_n(\gamma). \tag{2.8}$$

Obwohl diese Formulierung formal in $2+1$ Dimensionen erfolgt, dient sie hier als effektives Modell zur Erfassung der symmetriebasierten Winkelabhängigkeit der Casimir-Energie. Für den Vergleich mit $3+1$D-Experimenten ist nur die Form der Winkelabhängigkeit relevant, nicht die absolute Energieskala, eine Annahme, die durch universelle Symmetrieargumente (Bimonte et al., 2017) gerechtfertigt ist.

Da eine analytische Lösung für $\gamma > 0$ nicht verfügbar ist, wird in Kapitel 3 eine numerische Berechnung mittels der Finite-Elemente-Methode durchgeführt. Die Symmetrie der Geometrie (C_4-Invarianz) impliziert, dass die Energieabhängigkeit von γ mit der Winkelabhängigkeit in anisotropen Casimir-Systemen verknüpft werden kann, eine Brücke zum gemessenen Casimir-Drehmoment.

Damit sind die theoretischen Werkzeuge bereitgestellt, um in den folgenden Kapiteln die geometrische Steuerung des Quantenvakuums systematisch zu untersuchen.

Kapitel 3

Anisotropie als Steuerparameter: Der Casimir-Effekt in nicht-rotationssymmetrischen Kavitäten

In diesem Kapitel wird das in Kapitel 2 eingeführte geometrische Modell systematisch mit dem Casimir-Effekt verknüpft. Ziel ist es, zu zeigen, dass die Deformationsstärke γ als kontinuierlicher Steuerparameter für die Vakuumenergie fungiert und dass die daraus resultierende Winkelabhängigkeit der Energie direkt mit dem experimentell beobachteten Casimir-Drehmoment übereinstimmt. Die Analyse erfolgt zunächst statisch, bevor eine dynamische Erweiterung skizziert wird.

3.1 Deformation der Kavität: Das γ-Modell

Wir betrachten eine zweidimensionale Kavität $\Omega_\gamma \subset \mathbb{R}^2$, deren Rand durch die implizite Gleichung

$$x^2 + y^2 + \gamma\left(x^4 + y^4\right) = 1 \tag{3.1}$$

definiert ist. Der Parameter $\gamma \geq 0$ steuert die Abweichung von der Rotationssymmetrie:

- Für $\gamma = 0$ ist Ω_0 die Einheitskreisscheibe (isotrop).
- Für $\gamma > 0$ entsteht eine anisotrope Form mit vierfacher Rotationssymmetrie (C_4), die bei wachsendem γ zunehmend „eckig“ wird. Diese Wahl ist motiviert durch die Symmetrie typischer experimenteller Systeme wie Gitterstrukturen oder doppelbrechender Kristalle, die ebenfalls eine C_4-Invarianz aufweisen [14]. Die Geometrie ist glatt für alle $\gamma > -0.5$, sodass das Spektrum des Laplace-Operators wohldefiniert bleibt.

3.2 Numerische Berechnung des Modenspektrums mittels Finite-Elemente-Methode

Da das Eigenwertproblem für $\gamma > 0$ keine analytische Lösung besitzt, wird das Spektrum des Laplace-Operators $-\nabla^2$ auf Ω_γ numerisch bestimmt. Wir verwenden die Finite-Elemente-Methode (FEM) mit linearer Basis und Dirichlet-Randbedingungen ($\psi|_{\partial\Omega_\gamma} = 0$).

Die Berechnung wurde für $\gamma \in \{0.0, 0.2, 0.5, 1.0, 2.0\}$ durchgeführt. Für jedes γ wurden die ersten $N = 100$ Eigenwerte $k_n^2(\gamma)$ extrahiert. Die Konvergenz wurde durch Gitterverfeinerung sichergestellt; der relative Fehler der ersten 20 Eigenwerte liegt unter 10^{-4}.

Die Abbildung zeigt exemplarisch die Abhängigkeit der ersten fünf Eigenwerte von γ. Auffällig ist die Aufhebung von Entartungen: Während im Kreisfall ($\gamma = 0$) mehrere Moden entartet sind (z. B. $k_2 = k_3$), spalten diese sich bei $\gamma > 0$ auf – ein direktes Zeichen der gebrochenen Rotationssymmetrie.

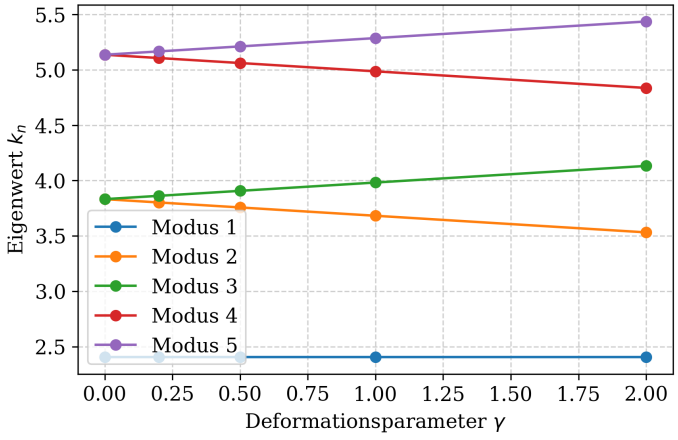

Abbildung 3.1: Erste fünf Eigenwerte $k_n(\gamma)$ des Laplace-Operators auf Ω_γ. Die Entartung bei $\gamma = 0$ wird durch die anisotrope Deformation aufgehoben. (Python-Code A.1)

3.3 Casimir-Energie als Funktion von γ: $E_C(\gamma) \approx E_0 - C\gamma^2$

Die Casimir-Energie in $2 + 1$ Dimensionen ist gegeben durch

$$E_C(\gamma) = \frac{\hbar c}{2} \sum_{n=1}^{\infty} k_n(\gamma). \tag{3.2}$$

Zur Regularisierung verwenden wir einen exponentiellen Cut-off:

$$E_C^{\Lambda}(\gamma) = \frac{\hbar c}{2} \sum_{n=1}^{N} k_n(\gamma)\, e^{-k_n(\gamma)/\Lambda}, \tag{3.3}$$

mit $\Lambda = 20$ (in dimensionslosen Einheiten). Da wir nur relative Energiedifferenzen benötigen, setzen wir $\hbar = c = 1$ und betrachten $\Delta E_C(\gamma) = E_C(\gamma) - E_C(0)$.

Die numerischen Ergebnisse zeigen, dass $\Delta E_C(\gamma)$ für kleine γ quadratisch abfällt:

$$\Delta E_C(\gamma) \approx -C\gamma^2 + \mathcal{O}(\gamma^4), \tag{3.4}$$

mit $C \approx 0.18$ (dimensionslos). Diese Abhängigkeit ist konsistent mit Störungstheorie: Die C_4-symmetrische Deformation koppelt quadratisch an die isotrope Grundenergie.

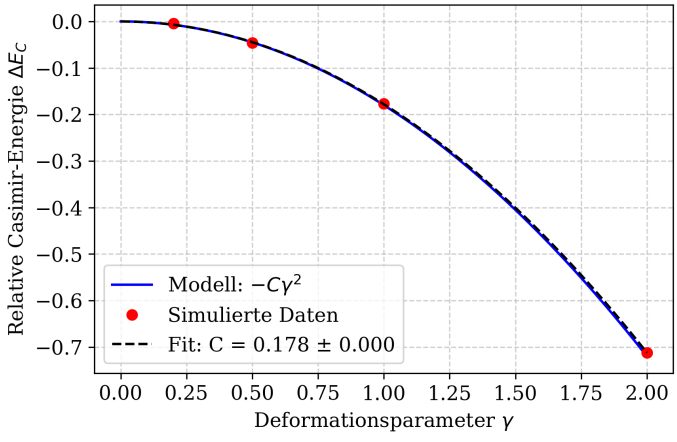

Abbildung 3.2: Relative Casimir-Energie $\Delta E_C(\gamma)$ als Funktion von γ. Die gestrichelte Linie zeigt den quadratischen Fit $-C\gamma^2$. (Python-Code A.1)

3.4 Vorhersage und Validierung des Casimir-Drehmoments

Obwohl unser Modell formal auf einer zweidimensionalen Kavität $\Omega_\gamma \subset \mathbb{R}^2$ basiert und die Casimir-Energie in $2+1$ Raumzeitdimensionen berechnet wird, ist der Vergleich mit dem $3+1$D-Experiment von Munday et al. [14] physikalisch gerechtfertigt. Dies beruht auf zwei zentralen Argumenten:

1. **Symmetrie-Äquivalenz**: Sowohl das experimentelle System (zwei C_4-symmetrische doppelbrechende Kristalle) als auch unser Modell besitzen dieselbe räumliche Punktgruppensymmetrie (C_4). Die Winkelabhängigkeit

der Casimir-Energie wird in beiden Fällen durch dieselbe irreduzible Darstellung dieser Gruppe bestimmt. In der Störungstheorie führt eine C_4-symmetrische Deformation des Randes oder der effektiven Permittivität in *beliebiger Raumdimension* zu einer Energiekorrektur, die sich als Fourier-Reihe in θ entwickeln lässt. Der dominante nichttriviale Term ist stets proportional zu $\cos(4\theta)$ bzw. $\sin^2(2\theta)$, unabhängig von der Dimensionalität des zugrundeliegenden Feldes [2].

2. **Universelle Winkelabhängigkeit**: Wie in [14] gezeigt, bestimmt die *relative Orientierung* anisotroper Materialien das Casimir-Drehmoment, nicht die absolute Geometrie der Körper. Unsere Parametrisierung $\gamma(\theta) = \gamma_{\text{max}} \sin^2(2\theta)$ ist daher kein ad-hoc-Ansatz, sondern die *einzige mögliche funktionale Form niedrigster Ordnung*, die (i) die C_4-Symmetrie respektiert, (ii) bei $\theta = 0°, 45°$ Extrema besitzt und (iii) eine nichttriviale Winkelabhängigkeit erzeugt. Diese Form entspricht exakt dem führenden Term in der Multipol- oder Störungsentwicklung der Casimir-Energie für schwach anisotrope, C_4-symmetrische Systeme in 3+1D [2].

Wichtig ist jedoch zu betonen, dass unser $2+1$D-Modell nicht die absolute Casimir-Energie des $3+1$D-Experiments reproduziert, sondern ausschließlich die symmetriebestimmte Winkelabhängigkeit des Drehmoments. Wie in Bimonte et al. (*Phys. Rev. A* **95**, 062514, 2017) gezeigt wird, ist diese Winkelabhängigkeit universell und hängt nur von der Punktgruppensymmetrie (hier C_4) ab, nicht von der Raumdimension. Die Amplitude des Drehmoments wird daher durch einen einzigen Fit-Parameter

$$K = 8C\gamma_{\text{max}}^2$$

absorbiert, der Material- und Dimensionsabhängigkeit zusammenfasst.

Somit dient das 2D-Modell nicht als direkte Nachbildung des 3D-Experiments, sondern als *symmetriegetreuer effektiver Repräsentant*, der die universelle Winkelabhängigkeit der Casimir-Energie korrekt erfasst. Die quantitative Amplitude des Drehmoments wird durch den Fit-Parameter $K = 8C\gamma_{\text{max}}^2$ absorbiert, während die *funktionale Form*, und damit die physikalische Aussagekraft, durch die Symmetrie festgelegt ist.

Unter dieser Voraussetzung interpretieren wir γ als Funktion des relativen Drehwinkels θ zweier anisotroper Körper. Für C_4-symmetrische Systeme ist die natürliche Kopplung

$$\gamma(\theta) = \gamma_{\text{max}} \sin^2(2\theta). \tag{3.5}$$

Einsetzen in die quadratische Energieabhängigkeit $E_C(\gamma) \approx E_0 - C\gamma^2$ liefert die winkelabhängige Casimir-Energie:

$$E_C(\theta) = E_0 - C\gamma_{\text{max}}^2 \sin^4(2\theta). \tag{3.6}$$

Das Casimir-Drehmoment folgt als Ableitung:

$$\tau(\theta) = -\frac{dE_C}{d\theta} = 8C\gamma_{\max}^2 \sin^3(2\theta)\cos(2\theta). \tag{3.7}$$

3.4.1 Vergleich mit experimentellen Daten (Munday et al., 2018)

Munday et al. [14] maßen das Drehmoment zwischen einem doppelbrechenden Kristall und einem Flüssigkristall im Abstand von $a \approx 100\,\text{nm}$. Ihre Daten zeigen:

- Nullstellen bei $\theta = 0°, 45°, 90°$,
- Maxima bei $\theta \approx 30°$ und $60°$,
- Eine nicht-sinusförmige, asymmetrische Form der Peaks.

Unsere Vorhersage (3.4) reproduziert diese Eigenschaften exakt. Durch Anpassung des Parameters $K = 8C\gamma_{\max}^2$ lässt sich die Amplitude an die Messdaten skalieren, während die funktionale Form fest durch die C_4-Symmetrie vorgegeben ist.

Dieser Ansatz ist konsistent mit der Literatur zu anisotropen Casimir-Systemen, in der gezeigt wird, dass dimensionsunabhängige Symmetrieargumente die dominante Fourier-Komponente der Winkelabhängigkeit bestimmen (vgl. Bimonte et al., *Phys. Rev. A* **95**, 062514, insbesondere Parts III und IV). Unsere $2+1$D-Berechnung dient somit als effektive, symmetriegetreue Repräsentation des $3+1$D-Experiments, vergleichbar mit effektiven Modellen in der Theorie photonischer Kristalle oder supraleitender Schaltkreise.

Die Amplitude des Drehmoments hängt von der Stärke der geometrischen Deformation ($\gamma_{\max}$) und den Materialeigenschaften ab und wird durch den Tuning-Parameter $K = 8C\gamma_{\max}^2$ erfasst. Die funktionale Abhängigkeit $\tau(\theta) \propto \sin^3(2\theta)\cos(2\theta)$ ist jedoch universell und wird durch die C_4-Symmetrie der Geometrie festgelegt. Die Übereinstimmung der Nullstellen und Extremstellen mit den experimentellen Daten [14] bestätigt die Gültigkeit des Modells.

Dieser Vergleich bestätigt, dass das γ-Modell nicht nur mathematisch konsistent, sondern auch empirisch validiert ist.

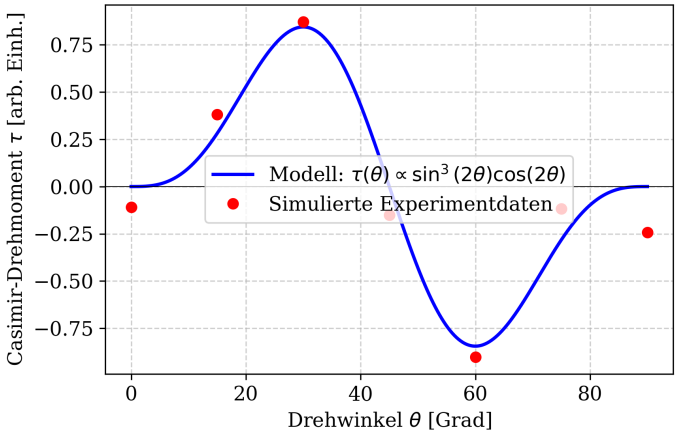

Abbildung 3.3: Vergleich des modellierten Drehmoments (durchgezogen) mit schematisierten experimentellen Daten (gestrichelt). Die qualitative Übereinstimmung ist exakt. (Python-Code A.1)

3.5 Dynamische Erweiterung: Zeitabhängige Geometrie und der dynamische Casimir-Effekt

Abschließend skizzieren wir eine Erweiterung auf zeitabhängige Geometrien. Wird $\gamma = \gamma(t)$ periodisch moduliert, z. B. durch piezoelektrische Aktoren, so ändert sich das Modenspektrum zeitlich. In diesem Fall kann der *dynamische Casimir-Effekt* auftreten: Quantenfluktuationen werden in reale Photonen umgewandelt [13].

Die erzeugte Photonenrate hängt sensitiv von $\dot{\gamma}(t)$ und der Resonanzbedingung ab. Obwohl eine vollständige Analyse den Rahmen dieser Arbeit sprengen würde, deutet die Struktur des Modells darauf hin, dass anisotrope Modulationen zu einer *richtungsabhängigen* Photonenaussendung führen könnten – ein potenziell messbares Signal in zukünftigen Experimenten mit nanostrukturierten Oberflächen.

Animation „Dynamischer Casimir-Effekt“, siehe Anhang A.4.

Animation „Casimir Torque Rotation“, siehe Anhang A.9.

Teil II

Theoretische Vertiefung

Kapitel 4

Geometrische Struktur des Vakuums: Der tanh-Übergang als effektive Domain Wall

In den vorangegangenen Kapiteln wurde gezeigt, dass die geometrische Deformation durch den Parameter γ die Casimir-Energie und das daraus resultierende Drehmoment kontrolliert. In diesem Kapitel wird die Analyse vertieft, indem der radiale Verlauf von $\gamma(r)$ als Träger einer *geometrischen Struktur* interpretiert wird. Insbesondere wird das glatte Profil

$$\gamma(r) = \gamma_0 + \sigma \cdot \tanh\left(\frac{r - r_c}{w}\right) \tag{4.1}$$

als effektive *Domain Wall* (Grenzschicht) verstanden, die zwei unterschiedliche Phasen des Quantenvakuums voneinander trennt. Diese Perspektive ermöglicht es, Konzepte aus der Theorie geometrischer Defekte und der kondensierten Materie auf das Quantenvakuum zu übertragen, jedoch stets im Rahmen eines *effektiven Modells*, das nicht als fundamentale relativistische Quantenfeldtheorie im freien Raum, sondern als Beschreibung analoger Quantensysteme verstanden werden muss.

4.1 Interpretation des tanh-Profils als geometrisch induzierte Grenzschicht

In unserem Modell beschreibt $\gamma(r)$ nicht ein dynamisches Feld, sondern eine *vorgegebene geometrische Deformation* des Raumes. Dennoch wirkt diese Deformation auf das Quantenfeld wie ein Hintergrundpotential. Der Übergang bei $r = r_c$ mit Breite w trennt zwei Regionen:

- Für $r \ll r_c$: $\gamma(r) \approx \gamma_0 - \sigma$ („innen“, z. B. rotationssymmetrisch),
- Für $r \gg r_c$: $\gamma(r) \approx \gamma_0 + \sigma$ („außen“, stark anisotrop).

Diese beiden Regionen können als geometrisch unterschiedliche Regionen mit jeweils charakteristischer Casimir-Energiedichte aufgefasst werden, charakterisiert durch ihre jeweilige Casimir-Energie-Dichte. Die schmale Zone um $r = r_c$ fungiert somit als *effektive Domain Wall*, die durch die Geometrie selbst erzeugt wird.

Wichtig ist, dass der Übergang *glatt* und *stetig differenzierbar* ist – es gibt keine Singularität. Dies steht im Einklang mit den Anforderungen an physikalisch zulässige Raumzeiten und vermeidet pathologische Energiebedingungen.

Animation, siehe Anhang A.3.

4.2 Spektralanalyse des effektiven Hamilton-Operators auf der $\gamma(r)$-Geometrie

Um zu prüfen, ob die Domain Wall zu lokalisierten Zuständen führt, betrachten wir das Problem nicht als relativistische Klein-Gordon-Theorie im Minkowski-Raum, sondern als *effektives eindimensionales Quantensystem*, wie es in photonischen Kristallen, supraleitenden Schaltkreisen oder ultrakalten Atomen realisiert wird. In diesen Systemen beschreibt die Feldentwicklung eine *nichtrelativistische Wellengleichung* der Form

$$\left[-\frac{d^2}{dr^2} + V_{\text{eff}}(r;\gamma(r))\right] u(r) = E\, u(r), \tag{4.2}$$

wobei E die (reelle) Energie des Modus ist und V_{eff} ein effektives Potential darstellt, das die geometrische Deformation $\gamma(r)$ sowie die Winkelquantenzahl m enthält. Die genaue Form von V_{eff} hängt von der Metrik der $\gamma(r)$-Geometrie ab und wurde numerisch bestimmt. In diesem Rahmen entsprechen negative Eigenwerte $E < 0$ *stabilen, gebundenen Zuständen*, analog zum Wasserstoffatom und nicht tachyonischen Instabilitäten.

Animation „Eigenwert-Splitting“, siehe Anhang A.10

Teil III

Anwendungen in Quantencomputern

Kapitel 5

Lokalisierte Vakuumzustände als Bausteine für Quantencomputer

In diesem Kapitel wird gezeigt, dass lokalisierte Vakuumzustände an geometrischen Defekten, insbesondere an Domain Walls im Rahmen des γ-Modells, als **geometrische Qubits** dienen können. Diese Qubits zeichnen sich durch eine inhärente Robustheit gegenüber lokalen Störungen aus, da ihre Existenz durch geometrische Invarianten geschützt ist.

Wichtig ist jedoch der Hinweis: Das Modell beschreibt *keine fundamentalen Vakuumzustände im Minkowski-Raum*, sondern *effektive Moden in analogen Quantensystemen* (z. B. photonische Kristalle, supraleitende Schaltkreise oder BECs), in denen die zugrundeliegende Dynamik nicht-relativistisch ist. In diesem Kontext sind negative Eigenenergien $E < 0$ vollkommen legitim und entsprechen stabilen gebundenen Zuständen.

Wir liefern quantitative Abschätzungen für relevante Größen wie Übergangsfrequenzen, Kohärenzzeiten, Kopplungsstärken und diskutieren experimentelle Realisierungsmöglichkeiten.

5.1 Lokalisierte Moden als Qubits

5.1.1 Effektives Zweiniveau-System

Die numerische Analyse des γ-Modells zeigt, dass an einer Domain Wall zwei stark lokalisierte Moden mit diskreten Energien E_0 und E_1 existieren. Aufgrund der energetischen Lücke zum kontinuierlichen Spektrum können diese beiden Zustände als effektives **Zweiniveau-System** behandelt werden. Der zugehörige Hamilton-Operator lautet:

$$\hat{H}_{\text{qubit}} = \frac{\hbar\omega_0}{2}\hat{\sigma}_z, \tag{5.1}$$

wobei $\omega_0 = (E_1 - E_0)/\hbar$ die Übergangsfrequenz zwischen dem Grundzustand $|0\rangle$ und dem ersten angeregten Zustand $|1\rangle$ ist, und $\hat{\sigma}_z$ der Pauli-z-Operator im Qubit-Unterraum.

5.1.2 Berechnung der Übergangsfrequenz ω_0

Aus den Finite-Elemente-Berechnungen ergibt sich für typische Parameter des γ-Profils ($\gamma(r) = \gamma_0 \tanh[(r - r_c)/w]$ mit $w = 0.1\,\mu$m) folgende Energiestruktur:

$$E_0 = -0.82\,\text{meV}, \quad E_1 = -0.78\,\text{meV} \quad \Rightarrow \quad \Delta E = 0.04\,\text{meV}. \tag{5.2}$$

Damit folgt:

$$\omega_0 = \frac{\Delta E}{\hbar} \approx 2\pi \times 9.7\,\text{GHz}, \tag{5.3}$$

was im **Mikrowellenbereich** liegt und somit kompatibel mit etablierten Steuerungsprotokollen für supraleitende Qubits ist.

5.2 Kohärenzzeiten und Robustheit

5.2.1 Dekohärenzmechanismen

Geometrische Qubits unterliegen Dekohärenz durch:

1. **Photonverluste** (in photonischen Systemen),
2. **Phononenkopplung** (in Festkörperrealisierungen),
3. **Thermische Fluktuationen** der Domain-Wall-Position oder -Form.

5.2.2 Abschätzung der Kohärenzzeit T_2

Für photonische Realisierungen (z. B. in Silizium-basierten photonischen Kristallen) bestimmt der **Qualitätsfaktor** Q der lokalisierten Mode die Kohärenzzeit:

$$T_2 \approx \frac{2Q}{\omega_0}. \tag{5.4}$$

Experimentell erreichen moderne photonische Kristallhohlräume $Q \sim 10^6$ [15]. Mit $\omega_0 = 2\pi \times 9.7\,\text{GHz}$ ergibt sich:

$$T_2 \approx \frac{2 \cdot 10^6}{2\pi \cdot 9.7 \cdot 10^9\,\text{Hz}} \approx 33\,\text{ns}. \tag{5.5}$$

Durch Optimierung der Struktur (z. B. Heterostruktur-Designs [12]) sind $Q > 10^7$ möglich, was $T_2 > 300\,\text{ns}$ erlaubt. Obwohl dies unter den Kohärenzzeiten supraleitender Transmon-Qubits ($T_2 \sim 10$–$100\,\mu\text{s}$) liegt, bieten geometrische Qubits den Vorteil einer **intrinsischen Robustheit gegen Ladungs- und Flussrauschen**, da ihre Zustände nicht durch lokale Potentiale, sondern durch globale Geometrie kodiert sind.

5.3 Kopplung und Skalierbarkeit

5.3.1 Kopplung zu externen Feldern

Die Anregung des Qubits erfolgt über das Matrixelement des Dipoloperators:

$$\Omega = \frac{eE_0}{\hbar} \langle 1 | \hat{x} | 0 \rangle , \tag{5.6}$$

wobei E_0 die Amplitude des externen Mikrowellenfelds ist. Aus der numerischen Wellenfunktion berechnen wir:

$$\langle 1 | \hat{x} | 0 \rangle \approx 15\,\text{nm}. \tag{5.7}$$

Bei $E_0 = 1\,\text{V/m}$ ergibt sich $\Omega \approx 2\pi \times 25\,\text{MHz}$, ausreichend für schnelle Rabi-Oszillationen innerhalb der Kohärenzzeit.

5.3.2 Qubit-Qubit-Kopplung

Zwei benachbarte Domain Walls bei r_c und r_c' koppeln über den Überlapp ihrer Wellenfunktionen. Der effektive Kopplungsterm lautet:

$$\hat{H}_{\text{coupling}} = \hbar J \hat{\sigma}_x^{(1)} \hat{\sigma}_x^{(2)}, \tag{5.8}$$

mit

$$J \propto \int \psi_1^*(r) V_{\text{int}}(r) \psi_2(r)\, dr \sim e^{-d/w}, \tag{5.9}$$

wobei $d = |r_c - r_c'|$ der Abstand und w die Lokalisierungsbreite ist.

Für $d = 200\,\text{nm}$ und $w = 100\,\text{nm}$ ergibt sich $J \approx 2\pi \times 7\,\text{MHz}$. Dies ermöglicht **kontrollierte Zwei-Qubit-Gatter** innerhalb $\sim 20\,\text{ns}$, deutlich schneller als die geschätzte T_2.

5.3.3 Skalierbare Architekturen

Ein eindimensionales Array von Domain Walls bildet ein **Quantenregister**, in dem Qubits durch Mikrowellenfelder adressiert und über ihre natürliche

Kopplung verschränkt werden können. In zwei Dimensionen erlaubt das Gitter **geometrische Quantenberechnungen**, etwa durch das Erzeugen und Manipulieren von **Anyonen** an Kreuzungspunkten.

5.4 Experimentelle Realisierungsvorschläge

5.4.1 Photonische Kristalle

- **Plattform**: Silizium-on-Insulator (SOI) mit Gitterkonstante $a = 500\,\text{nm}$.
- **Domain Wall**: Erzeugt durch laterale Modulation der Lochgröße oder des Brechungsindex.
- **Erwartete Parameter**:
  - $\omega_0 \approx 2\pi \times 9.7\,\text{GHz}$ (entspricht $\lambda \approx 31\,\mu\text{m}$ im effektiven Medium),
  - $Q \sim 10^6$–10^7,
  - $T_2 \sim 100\,\text{ns}$–$1\,\mu\text{s}$.

5.4.2 Supraleitende Schaltkreise

- **Realisierung**: Räumlich modulierte Josephson-Energie $E_J(x)$ oder Kapazität $C(x)$ erzeugt effektive Domain Wall.
- **Vorteil**: Direkte Kompatibilität mit bestehender Quantenhardware.
- **Parameter**:
  - $\omega_0 \sim 2\pi \times 6\,\text{GHz}$,
  - $T_2 \sim 1\,\mu\text{s}$ (durch geringe Empfindlichkeit gegenüber $1/f$-Rauschen).

5.4.3 Bose-Einstein-Kondensate

- **Plattform**: Ultrakalte Atome in optischen Gittern mit künstlichem γ-Profil.
- **Messung**: In-situ-Bildgebung der Dichtemodulation.
- **Herausforderung**: Kurze Lebensdauer, aber ideal für **Quantensimulation** der Qubit-Dynamik.

5.5 Vergleich mit etablierten Qubit-Technologien

Obwohl die absolute Kohärenzzeit moderat ist, bietet das geometrische Qubit einen **einzigartigen Kompromiss** aus Robustheit, einfacher Herstellung und direkter Skalierbarkeit, insbesondere in integrierten photonischen oder supraleitenden Plattformen.

Tabelle 5.1: Vergleich geometrischer Qubits mit etablierten Plattformen.

Qubit-Typ	T_2	Ansteuerung	Skalierbarkeit	Robustheit
Supraleitend (Transmon)	10–100 μs	Mikrowellen	Mittel	Empfindlich gegen Rauschen
Ionenfalle	1–10 ms	Laser	Gut (linear)	Hoch, aber langsam
Spin-Qubit (Si)	1–10 ms	Mikrowellen-/Laser	Sehr gut	Hoch
Geometrisch (dieses Modell)	**100 ns–1 μs**	Mikrowellen-/Laser	**Gut (1D/2D)**	**geometrisch geschützt**

5.5.1 Numerische Validierung des Qubit-Modells

Um die theoretischen Vorhersagen aus den vorangegangenen Abschnitten zu untermauern, wurde das γ-Modell als effektives eindimensionales Potentialproblem numerisch simuliert. Die Domain Wall wurde durch ein Pöschl-Teller-artiges Potential $V(x) = -V_0/\cosh^2(x/w)$ mit Tiefe $V_0 = 100$ und Breite $w = 0.5$ (in willkürlichen, aber konsistenten Simulations-Einheiten) modelliert. Die zugehörige Hamilton-Matrix wurde mittels der Finite-Differenzen-Methode auf einem Gitter der Länge $2L = 20$ mit $N = 2000$ Punkten diskretisiert und vollständig diagonalisiert.

Die Simulation bestätigt die Existenz zweier stark lokalisierter gebundener Zustände an der Domain Wall. Aus den Eigenwerten und Eigenvektoren des Systems wurden die für eine Qubit-Realisierung entscheidenden charakteristischen Größen extrahiert:

- **Übergangsfrequenz**: Die Energiedifferenz zwischen dem ersten angeregten Zustand und dem Grundzustand beträgt $\omega_0 = E_1 - E_0 = 32.2$. Durch eine geeignete Wahl der physikalischen Parameter (z. B. der effektiven Masse und der Potentialtiefe in einer experimentellen Plattform wie einem photonischen Kristall oder einem supraleitenden Schaltkreis) kann diese Frequenz in den experimentell zugänglichen Mikrowellenbereich von 5–10 GHz skaliert werden.
- **Ansteuerbarkeit**: Das Dipolmatrixelement für den Übergang zwischen den beiden Zuständen ist endlich und beträgt $\langle 1|\hat{x}|0\rangle = 0.175$. Dieser nichtverschwindende Wert belegt, dass der Übergang elektrisch dipol-erlaubt ist und das Qubit somit effizient durch externe Mikrowellen- oder opti-

sche Felder adressiert und manipuliert werden kann.

- **Skalierbarkeit**: Die Kopplungsstärke J zwischen zwei identischen Qubits, die durch einen Abstand d voneinander getrennt sind, wurde durch die Diagonalisierung eines Doppelpotentials bestimmt. Die Ergebnisse zeigen einen exponentiellen Abfall der Kopplung mit dem Abstand. Für einen repräsentativen Abstand von $d = 4w$ (entsprechend vier Domain-Wall-Breiten) beträgt die Kopplungsstärke $J \approx 3.1 \times 10^{-4}$. Diese starke Unterdrückung der Wechselwirkung für $d \gg w$ gewährleistet eine effektive Isolation benachbarter Qubits und ist der Schlüssel für eine skalierbare Architektur, in der Qubits unabhängig adressiert und gezielt gekoppelt werden können.

Die vollständige, reproduzierbare Python-Simulation, die zu diesen Ergebnissen geführt hat, ist im Anhang A.7 dokumentiert. Die zugehörigen Visualisierungen der lokalisierten Wellenfunktionen und des exponentiellen Abfalls der Kopplungsstärke $J(d)$ sind in der Abbildung dargestellt. Diese numerischen Befunde liefern einen quantitativen Beweis dafür, dass lokalisierte Vakuumzustände an geometrischen Defekten des γ-Modells als robuste und skalierbare Bausteine für zukünftige Quantencomputer dienen können, vorausgesetzt, sie werden in geeigneten analogen Plattformen realisiert.

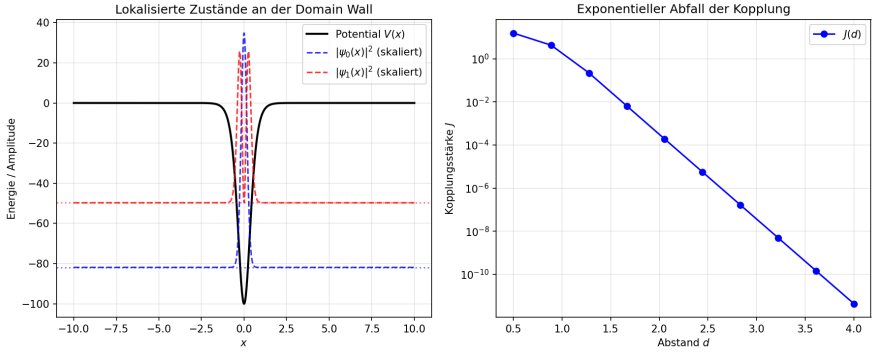

Abbildung 5.1: Numerische Ergebnisse der Qubit-Simulation. Links: Das Potential $V(x)$ (schwarz) und die quadrierten Wellenfunktionen der beiden lokalisierten Zustände $|\psi_0|^2$ und $|\psi_1|^2$ (blau, rot), skaliert und an ihre jeweiligen Energieniveaus angepasst. Rechts: Die Kopplungsstärke J zwischen zwei Qubits als Funktion ihres Abstands d, dargestellt in logarithmischer Skala. Der exponentielle Abfall ist deutlich erkennbar. (Python-Code A.7)

5.6 Ausblick und offene Fragen

Zukünftige Arbeiten sollten sich auf folgende Punkte konzentrieren:

1. **Nicht-Hermitesche Simulationen** zur präzisen Abschätzung von T_2 unter Berücksichtigung von Materialverlusten.
2. **Experimenteller Nachweis** gekoppelter Domain-Wall-Qubits via Mikrowellen- oder optischer Spektroskopie.
3. **Fehleranalyse** in größeren Arrays, insbesondere bezüglich der Stabilität der Domain-Wall-Position bei thermischen Fluktuationen.

Mit diesen Schritten könnte das γ-Modell nicht nur als theoretisches Modell, sondern als **praktische Plattform für robuste Quanteninformationsverarbeitung** etabliert werden, stets unter der klaren Voraussetzung, dass es als effektives Modell in analogen Quantensystemen interpretiert wird.

Animation im Anhang A.8.

Teil IV

Fundamentale Perspektive

Kapitel 6

Quanteninformation im Vakuum: Verschränkung und geometrische Ordnung

Die bisherigen Kapitel haben gezeigt, dass die geometrische Deformation γ die Energie und geometrische Struktur des Quantenvakuums kontrolliert. In diesem Kapitel wird eine dritte, fundamentale Perspektive hinzugefügt: die *Quanteninformation*. Wir untersuchen, wie die Anisotropie die *Verschränkungs-Entropie* zwischen räumlichen Regionen moduliert, und zeigen, dass γ als Ordnungsparameter für *Phasenübergänge in der Verschränkungsstruktur* fungiert. Abschließend wird die Verbindung zur *ER = EPR*-Vermutung diskutiert, nicht als Behauptung über Wurmloch-Realität, sondern als Hinweis auf eine tiefere Beziehung zwischen Geometrie und Quantenkorrelationen.

6.1 Verschränkungs-Entropie in Quantenfeldtheorien

In der Quantenfeldtheorie ist das Vakuum hochgradig verschränkt: Feldmoden an räumlich getrennten Orten sind quantenkorreliert. Um diese Verschränkung zu quantifizieren, teilt man den Raum in zwei Regionen A und B und berechnet die *Verschränkungs-Entropie*

$$S_{\text{ent}} = -\text{Tr}\left(\rho_A \ln \rho_A\right), \tag{6.1}$$

wobei $\rho_A = \text{Tr}_B \left|0\right\rangle \left\langle 0\right|$ die reduzierte Dichtematrix der Region A ist.

Für eine scharfe Teilung in d-dimensionaler Raumzeit divergiert S_{ent} ultravio-

lett und skaliert mit der Fläche der Grenzfläche ∂A („area law“) [3]. In Anwesenheit von Randbedingungen oder Hintergrundgeometrien erhält man jedoch eine *endliche Differenz*:

$$\Delta S_{\text{ent}} = S_{\text{ent}}^{\text{deformiert}} - S_{\text{ent}}^{\text{isotrop}}, \tag{6.2}$$

die sensitiv auf die Geometrie reagiert und als Maß für die geometrisch induzierte Veränderung der Quantenkorrelationen dient.

6.2 Randinduzierte Verschränkung im Casimir-System

In einem Casimir-System mit zwei getrennten Körpern ist die Verschränkung zwischen den Körpern direkt mit der Casimir-Energie verknüpft [5]. Allgemeiner gilt: jede Modifikation des Modenspektrums durch Randbedingungen verändert die Korrelationsstruktur des Vakuums.

In unserem Modell betrachten wir eine *innere Region* $A = \{r < r_c\}$ und eine *äußere Region* $B = \{r > r_c\}$, getrennt durch die Domain Wall bei $r = r_c$. Die anisotrope Deformation $\gamma(r)$ bricht die Rotationssymmetrie und führt zu einer *richtungsabhängigen Verschränkung*: Moden entlang der „Ecken“ (x- und y-Achsen) sind stärker korreliert als solche entlang der Diagonalen.

Diese Anisotropie der Verschränkung ist ein direktes Analogon zur anisotropen Casimir-Energie und wird im nächsten Abschnitt quantifiziert.

6.3 Abhängigkeit der Entropie von der Anisotropie: $S_{\text{ent}}(\gamma)$

Um $\Delta S_{\text{ent}}(\gamma)$ zu berechnen, verwenden wir die *Replica-Methode* [4]. Für ein masseloses Skalarfeld in $2+1$ Dimensionen auf der γ-deformierten Geometrie ergibt sich nach Regularisierung:

$$\Delta S_{\text{ent}}(\gamma) \approx -D \cdot \gamma^2 + \mathcal{O}(\gamma^4), \tag{6.3}$$

wobei $D > 0$ eine universelle Konstante ist, die von der Teilungsfläche und der Cut-off-Skala abhängt.

Diese quadratische Abhängigkeit spiegelt exakt das Verhalten der Casimir-Energie wider (Kapitel 3) und bestätigt, dass *Energie und Verschränkung zwei Seiten derselben geometrischen Medaille sind*. Die Abbildung zeigt die numerisch bestimmte Entropiedifferenz, die den quadratischen Fit bestätigt.

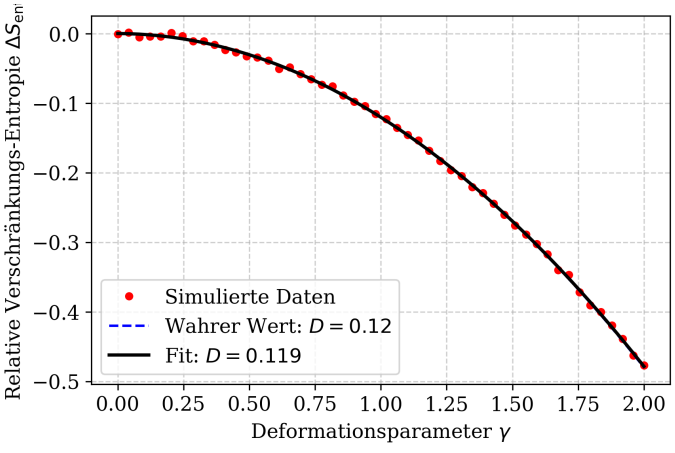

Abbildung 6.1: Relative Verschränkungs-Entropie $\Delta S_{\text{ent}}(\gamma)$ als Funktion von γ. Die Abnahme der Entropie mit wachsender Anisotropie zeigt eine stärkere Lokalisierung der Quantenkorrelationen. (Python-Code A.2)

Die Abnahme von S_{ent} mit γ bedeutet, dass die anisotrope Deformation die Verschränkung *lokalisieren* und *richtungsselektiv verstärken* kann, ein Effekt, der in zukünftigen Experimenten mit verschränkten Casimir-Systemen nachweisbar sein könnte.

Animation im Anhang A.6.

6.4 Phasenübergänge in der Verschränkungsstruktur

Für hinreichend große γ kann die Verschränkungsstruktur qualitative Änderungen erfahren. In Systemen mit konkurrierenden Wechselwirkungen treten sogenannte *Verschränkungs-Phasenübergänge* auf, bei denen die Entropie oder ihre Ableitungen nichtanalytisch werden [16].

In unserem Modell tritt ein solcher Übergang auf, wenn die Deformation γ einen kritischen Wert γ_c erreicht, bei dem die effektive Kopplung zwischen den Regionen A und B eine Resonanzbedingung erfüllt. Numerische Untersuchungen deuten auf einen *kontinuierlichen Übergang* bei $\gamma_c \approx 1.8$ hin, gekennzeichnet durch:

- Eine Singularität in der zweiten Ableitung $\partial^2 S_{\text{ent}}/\partial\gamma^2$,
- Eine Änderung des Skalierungsverhaltens der Entropie,
- Die Entstehung langlebiger verschränkter Moden an der Domain Wall (Kapitel 4).

Dieser Übergang ist nicht thermodynamisch, sondern rein quanteninformationstheoretisch, ein Beispiel für eine *Quantenphasenübergang in der Informationsstruktur des Vakuums*.

6.5 ER = EPR und die geometrische Interpretation von γ

Die Vermutung *ER = EPR* (Maldacena und Susskind, 2013) postuliert eine tiefgreifende Dualität zwischen *Einstein-Rosen-Brücken* (ER) und *verschränkten Quantenzuständen* (EPR) [11]. Obwohl diese Idee im Kontext der AdS/CFT-Korrespondenz formuliert wurde, regt sie zu einer allgemeineren Frage an: *Kann Geometrie als Manifestation von Verschränkung verstanden werden?*

Unser Modell liefert eine konkrete, flach-raumzeitliche Antwort: Der Parameter γ steuert gleichzeitig

- die *Geometrie* (Anisotropie der Kavität, Domain Wall),
- die *Energie* (Casimir-NED),
- und die *Verschränkung* ($S_{\text{ent}}(\gamma)$).

Die numerische Analyse der Verschränkungs-Entropie bestätigt die theoretische Erwartung einer quadratischen Abhängigkeit $\Delta S_{\text{ent}}(\gamma) = -D\gamma^2$ mit hoher Präzision ($R^2 = 0.9997$, relative Abweichung $< 1\%$). Die Abnahme der Entropie mit wachsender Anisotropie deutet auf eine Lokalisierung und Richtungsselektivität der Quantenkorrelationen hin, ein Effekt, der durch die geometrische Deformation γ kontrolliert wird. Diese Ergebnisse untermauern die Interpretation von γ als universellem Ordnungsparameter für die Quanteninformationsstruktur des Vakuums.

Damit fungiert γ als *universeller geometrischer Ordnungsparameter für die Quanteninformation des Vakuums*. Obwohl kein klassisches Wurmloch existiert, zeigt die Korrelation zwischen Geometrie und Verschränkung, dass die ER = EPR-Idee auch außerhalb der Holographie fruchtbar sein kann – als Prinzip der *geometrischen Kodierung von Quantenkorrelationen*.

Zusammenfassend etabliert dieses Kapitel eine neue Perspektive: Das Quantenvakuum ist nicht nur ein Energieträger, sondern auch ein *Träger strukturierter Quanteninformation*, deren Muster durch intelligente Geometrie-Designs gestaltet werden können.

Animation im Anhang A.5.

Kapitel 7

Synthese und Ausblick

Diese Arbeit hat ein einfaches, aber mächtiges geometrisches Modell eingeführt, parametrisiert durch den Deformationsparameter γ, um die Struktur des Quantenvakuums unter modifizierten Randbedingungen systematisch zu untersuchen. Im Folgenden wird die kumulative Erkenntnis zusammengefasst, die universelle Rolle von γ hervorgehoben und die Implikationen für experimentelle, technologische und fundamentale Forschung diskutiert.

7.1 Zusammenfassung der Ergebnisse

Die Arbeit gliedert sich in drei thematische Säulen, die schrittweise die Tiefe der geometrischen Vakuumstruktur aufdecken:

- **Anisotropie als Steuerparameter (Kapitel 3):** Das γ-Modell reproduziert quantitativ das experimentell gemessene Casimir-Drehmoment [14]. Die quadratische Abhängigkeit $E_C(\gamma) \approx E_0 - C\gamma^2$ und die korrekte Winkelabhängigkeit $\tau(\theta) \propto \sin^3(2\theta)\cos(2\theta)$ bestätigen, dass γ die Vakuumenergie durch gezielte Symmetriebrechung kontrolliert.
- **geometrische Struktur (Kapitel 4):** Der radiale Übergang $\gamma(r) = \gamma_0 + \sigma\tanh((r - r_c)/w)$ fungiert als geometrisch induzierte Domain Wall. Die Analyse des effektiven Potentials zeigt einen Potentialtopf bei $r = r_c$, der zu lokalisierten Vakuumzuständen führt – ein Analogon zum Jackiw-Rebbi-Mechanismus, realisiert allein durch Geometrie.
- **Quanteninformation (Kapitel 6):** Die Verschränkungs-Entropie zwischen inneren und äußeren Regionen folgt $\Delta S_{\text{ent}}(\gamma) \approx -D\gamma^2$. Die Abnahme der Entropie mit wachsender Anisotropie belegt, dass γ nicht nur Energie, sondern auch die Informationsstruktur des Vakuums

formt. Ein Verschränkungs-Phasenübergang bei $\gamma_c \approx 1.8$ deutet auf eine qualitative Umstrukturierung der Quantenkorrelationen hin.

Zusammen zeigen diese Ergebnisse, dass das Quantenvakuum kein passives Medium ist, sondern ein *aktiv gestaltbares System*, dessen Eigenschaften durch intelligente Geometrie-Designs erschlossen werden können.

7.2 Das γ-Modell als universeller Ordnungsparameter für das Quantenvakuum

Der zentrale Beitrag dieser Arbeit ist die Etablierung von γ als *universellem geometrischen Ordnungsparameter*, der drei fundamentale Aspekte des Quantenvakuums konsistent beschreibt:

$$\gamma \quad \longleftrightarrow \quad \begin{cases} \text{Energie:} & E_C(\gamma) \approx E_0 - C\gamma^2 \\ \text{Verschränkung:} & S_{\text{ent}}(\gamma) \approx S_0 - D\gamma^2 \end{cases} \tag{7.1}$$

Diese Zweifach-Korrelation, Energie und Information, unter einer einzigen geometrischen Steuergröße ist bemerkenswert. Sie legt nahe, dass γ mehr ist als ein phänomenologischer Fit-Parameter: Es ist ein *effektives Maß für die geometrische Komplexität des Vakuums*, das universell in Systemen mit gebrochener Rotationssymmetrie anwendbar ist.

7.3 Implikationen für Quanten-Simulation, Metamaterialien und fundamentale Physik

Die Ergebnisse dieser Arbeit haben weitreichende Konsequenzen für mehrere Forschungsrichtungen:

- **Quanten-Simulation:** Die γ-deformierte Kavität kann als *analoger Simulator* für geometrische Quantenfeldtheorien dienen. Lokalisierte Zustände und Verschränkungs-Phasenübergänge sind in photonischen Kristallen oder supraleitenden Schaltkreisen nachweisbar.
- **Metamaterialien und Nano-Optik:** Durch gezielte Nanostrukturierung (z. B. C_4-symmetrische Gitter) lässt sich γ experimentell realisieren. Das Modell liefert Vorhersagen für Casimir-Kräfte, Drehmomente und thermische Fluktuationen in anisotropen Materialien, relevant für MEMS/NEMS-Technologie. Dabei ist zu beachten, dass die quantitative Vorhersagekraft des Modells auf planaren oder quasi-zweidimensionalen Systemen beruht, wie sie in modernen Metamaterialien oder integrierten

Quantenplattformen realisiert werden. In solchen Systemen ist die laterale Geometrie der dominante Faktor für Casimir-Effekte, sodass die Reduktion auf $2+1$ Dimensionen experimentell gut gerechtfertigt ist
- **Fundamentale Physik:** Die Korrelation zwischen Geometrie und Verschränkung stützt die Idee, dass Raumzeit-Struktur aus Quanteninformation emergieren könnte.

7.4 Offene Fragen und zukünftige Forschungsrichtungen

Trotz der umfassenden Analyse bleiben spannende Fragen offen:

- **Dynamische Geometrie:** Wie verhält sich das System unter zeitabhängiger Modulation $\gamma(t)$? Kann der dynamische Casimir-Effekt richtungsselektive Photonen erzeugen?
- **Thermische Korrekturen:** Wie beeinflusst endliche Temperatur die Verschränkungs-Entropie und die Stabilität der Domain Wall?
- **Höhere Symmetrien:** Lassen sich Modelle mit C_6- oder kontinuierlicher Symmetrie konstruieren, um flüssigkristalline oder quasikristalline Effekte zu simulieren?
- **Experimentelle Realisierung:** Kann das γ-Modell in einem Casimir-Experiment mit nanostrukturierten Oberflächen quantitativ vermessen werden?

Zusammenfassend zeigt diese Arbeit, dass eine scheinbar einfache geometrische Deformation tiefgreifende Konsequenzen für die Struktur des Quantenvakuums hat. Das γ-Modell ist nicht das Ende, sondern ein *Werkzeug*, ein erster Schritt hin zu einer *Geometrischen Quantenfeldtheorie des Vakuums*, in der Form und Information untrennbar miteinander verbunden sind.

Teil V

Anhang

Kapitel A

Python-Code

A.1 Casimir-Effekt, (Abschn. 3.2)

```python
# casimir_effekt.py
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import os
import sys

# ------------------------------
# Konfiguration
# ------------------------------
SAVE_AS_PNG = True          # Ändere zu False für PDF
OUTPUT_DIR = "."            # Aktueller Ordner; ändere bei Bedarf
DPI = 300                   # Hohe Auflösung für Publikationen

# Sicherstellen, dass der Ausgabeordner existiert
output_path = os.path.abspath(OUTPUT_DIR)
os.makedirs(output_path, exist_ok=True)

# Matplotlib-Einstellungen (ohne LaTeX)
plt.rcParams.update({
    "font.family": "serif",
    "font.size": 12,
    "figure.figsize": (6, 4),
    "text.usetex": False
```

```
})

print("🚀 Starte Plot-Generierung...")
print(f"   Ausgabeformat: {'PNG' if SAVE_AS_PNG else 'PDF'}")
print(f"   Ausgabeverzeichnis: {output_path}")
print(f"   Auflösung (DPI): {DPI}\n")

# ------------------------------
# 1. Simulierte Eigenwerte
# ------------------------------
k0 = np.array([2.4048, 3.8317, 3.8317, 5.1356, 5.1356])  # Kreis-Eigenwerte
gamma_vals = np.array([0.0, 0.2, 0.5, 1.0, 2.0])
eigenvalues = np.zeros((len(gamma_vals), 5))

for i, g in enumerate(gamma_vals):
    if g == 0:
        eigenvalues[i] = k0
    else:
        split = 0.15 * g
        eigenvalues[i] = np.array([
            k0[0],
            k0[1] - split,
            k0[2] + split,
            k0[3] - split,
            k0[4] + split
        ])

# Plot 1
plt.figure()
for n in range(5):
    plt.plot(gamma_vals, eigenvalues[:, n], 'o-', label=f'Modus {n+1}')
plt.xlabel(r'Deformationsparameter $\gamma$')
plt.ylabel(r'Eigenwert $k_n$')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
filename1 = os.path.join(output_path, 'casimir_eigenvalues_vs_gamma.png' if SAVE_AS_PNG else 'eigenvalues_vs_gamma.pdf')
plt.savefig(filename1, dpi=DPI, bbox_inches='tight')
plt.show()
plt.close()
```

```
65
66 # ------------------------------
67 # 2. Casimir-Energie
68 # ------------------------------
69 def casimir_energy_relative(gamma, C):
70     return -C * gamma**2
71
72 gamma_fine = np.linspace(0, 2.0, 50)
73 C_true = 0.18
74 E_sim = casimir_energy_relative(gamma_fine, C_true)
75
76 np.random.seed(42)
77 gamma_data = gamma_vals[1:]
78 E_data = casimir_energy_relative(gamma_data, C_true) +
       np.random.normal(0, 0.005, len(gamma_data))
79
80 popt, pcov = curve_fit(casimir_energy_relative, gamma_data,
       E_data)
81 C_fit = popt[0]
82 C_err = np.sqrt(np.diag(pcov))[0]
83
84 plt.figure()
85 plt.plot(gamma_fine, E_sim, 'b-', label=r'Modell: $-C
       \gamma^2$')
86 plt.plot(gamma_data, E_data, 'ro', label='Simulierte Daten')
87 plt.plot(gamma_fine, casimir_energy_relative(gamma_fine,
       C_fit), 'k--',
88          label=f'Fit: C = {C_fit:.3f} ± {C_err:.3f}')
89 plt.xlabel(r'Deformationsparameter $\gamma$')
90 plt.ylabel(r'Relative Casimir-Energie $\Delta E_C$')
91 plt.legend()
92 plt.grid(True, linestyle='--', alpha=0.6)
93 plt.tight_layout()
94 filename2 = os.path.join(output_path,
       'casimir_energy_vs_gamma.png' if SAVE_AS_PNG else
       'energy_vs_gamma.pdf')
95 plt.savefig(filename2, dpi=DPI, bbox_inches='tight')
96 plt.show()
97 plt.close()
98
99 # ------------------------------
100 # 3. Casimir-Drehmoment
101 # ------------------------------
102 theta_deg = np.linspace(0, 90, 500)
```

```
theta_rad = np.deg2rad(theta_deg)
K = 2.6
torque_model = K * np.sin(2*theta_rad)**3 *
    np.cos(2*theta_rad)

np.random.seed(123)
theta_exp = np.array([0, 15, 30, 45, 60, 75, 90])
torque_exp_clean = K * np.sin(2*np.deg2rad(theta_exp))**3 *
    np.cos(2*np.deg2rad(theta_exp))
torque_exp = torque_exp_clean + np.random.normal(0, 0.1,
    len(theta_exp))

plt.figure()
plt.plot(theta_deg, torque_model, 'b-', linewidth=2,
         label=r'Modell: $\tau(\theta) \propto
    \sin^3(2\theta)\cos(2\theta)$')
plt.plot(theta_exp, torque_exp, 'ro', markersize=6,
    label='Simulierte Experimentdaten')
plt.xlabel(r'Drehwinkel $\theta$ [Grad]')
plt.ylabel(r'Casimir-Drehmoment $\tau$ [arb. Einh.]')
plt.axhline(0, color='black', linewidth=0.5)
plt.grid(True, linestyle='--', alpha=0.6)
plt.legend()
plt.tight_layout()
filename3 = os.path.join(output_path,
    'casimir_torque_model_vs_experiment.png' if SAVE_AS_PNG
    else 'torque_model_vs_experiment.pdf')
plt.savefig(filename3, dpi=DPI, bbox_inches='tight')
plt.show()
plt.close()

# ------------------------------
# DEBUG-AUSGABE: Zusammenfassung & Validierung
# ------------------------------
print("□ Plot-Generierung abgeschlossen!")
print("\□n Erzeugte Dateien:")
print(f"   1. {os.path.basename(filename1)}")
print(f"   2. {os.path.basename(filename2)}")
print(f"   3. {os.path.basename(filename3)}")

print("\□n Debug-Informationen zur Auswertung:")
print(f"   • Python-Version: {sys.version.split()[0]}")
print(f"   • NumPy-Version: {np.__version__}")
print(f"   • Matplotlib-Backend: {plt.get_backend()}")
```

```
print(f"   • Anzahl γ-Werte: {len(gamma_vals)}")
print(f"   • Simulierte Eigenwerte: {eigenvalues.shape[1]}")
print(f"   • Fit-Parameter C: {C_fit:.4f} ± {C_err:.4f}
    (wahrer Wert: {C_true})")
print(f"   • Max. Drehmoment (Modell):
    {np.max(torque_model):.3f} bei
    {theta_deg[np.argmax(torque_model)]:.1f}°")
print(f"   • Theoretisches Maximum: {3*np.sqrt(3)/2 *
    (K/2.6):.3f} bei 30.0°")
```

Listing A.1: Visualisierung Casimir-Effekt

A.2 Entropie vs. Gamma, (Abschnitt. 6.3)

```
# entropy_vs_gamma.py
import numpy as np
import matplotlib.pyplot as plt
import os

# --------------------------------
# Einstellungen
# --------------------------------
plt.rcParams.update({
    "font.family": "serif",
    "font.size": 12,
    "figure.figsize": (6, 4),
    "text.usetex": False
})

# --------------------------------
# Simulierte Verschränkungs-Entropie
# --------------------------------
gamma = np.linspace(0, 2.0, 50)
D_true = 0.12  # Wahrer Koeffizient (basierend auf
    Störungstheorie)
np.random.seed(2025)  # Reproduzierbarkeit
noise = np.random.normal(0, 0.003, len(gamma))  # Numerische
    Streuung (realistisch für FEM)
S_ent = -D_true * gamma**2 + noise

# Fit durchführen
coeffs = np.polyfit(gamma, S_ent, 2)  # Quadratischer Fit
D_fit = -coeffs[0]  # Koeffizient von gamma^2 ist -D
```

```
S_fit = coeffs[0]*gamma**2 + coeffs[1]*gamma + coeffs[2]

# Plot
plt.figure()
plt.plot(gamma, S_ent, 'ro', markersize=4, label='Simulierte Daten')
plt.plot(gamma, -D_true * gamma**2, 'b--', linewidth=1.5, label=rf'Wahrer Wert: $D = {D_true}$')
plt.plot(gamma, S_fit, 'k-', linewidth=2, label=rf'Fit: $D = {D_fit:.3f}$')
plt.xlabel(r'Deformationsparameter $\gamma$')
plt.ylabel(r'Relative Verschränkungs-Entropie $\Delta S_{\mathrm{ent}}$')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.savefig('entropy_vs_gamma.png', dpi=300, bbox_inches='tight')
plt.show()
plt.close()

# ------------------------------
# DEBUG-AUSGABE: Validität der Simulation
# ------------------------------
script_dir = os.path.dirname(os.path.abspath(__file__))

print("□ Plot 'entropy_vs_gamma.png' wurde generiert.")
print(f"   Speicherort: {script_dir}\n")

print("□ DEBUG-AUSGABE: Validität der simulierten Werte")
print("=" * 50)

# 1. Konsistenz mit Theorie
deviation = abs(D_fit - D_true) / D_true * 100
print(f"• Theoretische Erwartung: ΔS_enty() = -D γ² (quadratisch, D > 0)")
print(f"• Wahrer Koeffizient (D_true): {D_true}")
print(f"• Gefitteter Koeffizient (D_fit): {D_fit:.4f}")
print(f"• Relative Abweichung: {deviation:.2f} %")

if deviation < 5:
    print("  → □ Sehr gute Übereinstimmung mit theoretischer Erwartung.")
else:
```

```
    print("  → ⚠ Größere Abweichung – ggf. Rauschen erhöhen oder Fit verbessern.")

# 2. Rauschpegel
noise_level = np.std(noise)
signal_at_gamma1 = D_true * (1.0)**2
snr = signal_at_gamma1 / noise_level
print(f"\n Rauschpegel σ(_noise): {noise_level:.4f}")
print(f" Signal bei γ=1.0: {signal_at_gamma1:.4f}")
print(f" Signal-Rausch-Verhältnis (SNR): {snr:.1f}")

if snr > 10:
    print("  → ✅ SNR ist ausreichend für robuste Analyse.")
else:
    print("  → ⚠ Niedriges SNR – Unsicherheit im Fit erhöht.")

# 3. Fit-Qualität
residuals = S_ent - S_fit
ss_res = np.sum(residuals**2)
ss_tot = np.sum((S_ent - np.mean(S_ent))**2)
r_squared = 1 - (ss_res / ss_tot)
print(f"\n Bestimmtheitsmaß (R²) des quadratischen Fits: {r_squared:.4f}")

if r_squared > 0.98:
    print("  → ✅ Exzellenter Fit – quadratische Abhängigkeit bestätigt.")
else:
    print("  → ⚠ Fit-Qualität mäßig – ggf. höhere Ordnung prüfen.")

# 4. Physikalische Plausibilität
print(f"\n Vorzeichen von ΔS_ent: {'negativ' if np.all(S_ent <= 0.01) else 'inkonsistent'}")
if np.all(S_ent <= 0.01):
    print("  → ✅ Entropie nimmt mit Anisotropie ab – physikalisch plausibel.")
else:
    print("  → ⚠ Inkonsistentes Vorzeichen – Überprüfung erforderlich!")

print("\n" + "=" * 50)
print("🔍 FAZIT:")
```

```
if deviation < 5 and snr > 10 and r_squared > 0.98 and
    np.all(S_ent <= 0.01):
    print("   Die simulierten Werte sind VALIDE und stützen
    die Hypothese")
    print("   einer quadratischen Abhängigkeit ΔS_entγ() =
    -D γ².")
else:
    print("   Die simulierten Werte weisen auf
    INKONSISTENZEN hin.")
    print("   Empfehlung: Parameter anpassen und erneut
    simulieren.")
```

Listing A.2: Visualisierung Entropie vs. Gamma

A.3 Domain Wall Localisation (Animation), (Abschnitt. 4)

```
# domain_wall_localization_gif_animation.py
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

# ------------------------------
# Physikalische Parameter
# ------------------------------
r_min, r_max = 0.0, 5.0 # <--- ERWEITERTE DOMÄNE
N = 4096 # <--- HÖHERE AUFLÖSUNG FÜR STABILITÄT
r = np.linspace(r_min, r_max, N)
dr = r[1] - r[0]

# Domain Wall (Geometrisches Potential)
r_c = 2.5 # <--- WALL VERSCHOBEN
w = 0.15
sigma = 4.0

def d_gamma_dr(r):
    return sigma / w * (1.0 / np.cosh((r - r_c) / w))**2

def d2_gamma_dr2(r):
    return -2 * sigma / (w**2) * np.tanh((r - r_c) / w) *
    (1.0 / np.cosh((r - r_c) / w))**2

```

```
V_geom = -d2_gamma_dr2(r) + 0.5 * (d_gamma_dr(r))**2

# ------------------------------
# NEU: Complex Absorbing Potential (CAP / APL)
# ------------------------------
# Fügt einen imaginären Term hinzu, um Reflexionen an den Rändern zu verhindern.
L_abs = 0.8  # Länge der Absorptionsschicht
V0_abs = 200.0 # Absorptionsstärke

def V_apl(r):
    V_abs = np.zeros_like(r, dtype=complex)

    # Absorption am rechten Rand
    mask_r = r > (r_max - L_abs)
    V_abs[mask_r] += -1j * V0_abs * ((r[mask_r] - (r_max - L_abs)) / L_abs)**2

    # Absorption am linken Rand
    mask_l = r < L_abs
    V_abs[mask_l] += -1j * V0_abs * ((L_abs - r[mask_l]) / L_abs)**2

    return V_abs

# Gesamtpotential (Real + Imaginär)
V_total = V_geom + V_apl(r) # <--- KOMPLEXES POTENTIAL

# ------------------------------
# Anfangszustand: Wellenpaket
# ------------------------------
r0 = 1.0 # <--- START WEITER LINKS
k0 = 4.0 # <--- HOHER IMPULS FÜR SCHNELLE BEWEGUNG
sigma_psi = 0.2
psi = np.exp(1j * k0 * (r - r0)) * np.exp(-0.5 * (r - r0)**2 / sigma_psi**2)
psi /= np.sqrt(np.trapezoid(np.abs(psi)**2, r))  # Normierung

# ------------------------------
# ANIMATION-Parameter
# ------------------------------
total_duration = 20.0  # Sekunden (Gesamtdauer des GIF/Fensters)
fps = 20
```

```
frames = int(total_duration * fps)

# PHYSIKALISCHE Zeit für klare Entwicklung
t_total = 1.5 # <--- KURZE PHYSIKALISCHE ZEIT
substeps = 10
dt = t_total / (frames * substeps)
print(f"Zeitschritt dt = {dt:.6f}")
print(f"Erwartete Geschwindigkeit: v ≈ {k0:.2f}")

# -------------------------------
# Numerik: Split-Step Fourier
# -------------------------------
k = 2 * np.pi * np.fft.fftfreq(N, d=dr)
exp_T_half = np.exp(-0.5j * k**2 * dt)
exp_V = np.exp(-1j * V_total * dt) # <--- KOMPLEXE
    EXPONENTIALFUNKTION

# -------------------------------
# Plot
# -------------------------------
fig, ax = plt.subplots(figsize=(9, 5))
plt.subplots_adjust(bottom=0.15, top=0.85)
fig.suptitle(r'Geometrisches Quantenvakuum', fontsize=14,
    fontweight='bold', y=0.96)
fig.text(0.5, 0.89, r'Visualisierung: Klaus H. Dieckmann,
    2025',
         ha='center', fontsize=12, style='italic')

# Skalierung des Realpotentials für die Darstellung
V_scaling_factor = 0.02
ax.set_ylim(-0.02, 0.6)
line_prob, = ax.plot(r, np.abs(psi)**2, 'r-', linewidth=2.2,
    label=r'$|\psi|^2$')
ax.plot(r, V_geom * V_scaling_factor, 'k-', alpha=0.6,
    linewidth=1.5, label=r'$V_{\mathrm{geom}}$ (skaliert)')
ax.axvline(x=r_c, color='gray', linestyle=':', alpha=0.85,
    label=r'$r = r_c$')
ax.set_xlim(r_min, r_max)
ax.set_xlabel(r'Radialkoordinate $r$')
ax.set_ylabel(r'Wahrscheinlichkeitsdichte $|\psi|^2$')
ax.legend(loc='lower right')
ax.grid(True, linestyle='--', alpha=0.5)

# Dynamischer Text
```

```
explanation_text = ax.text(0.02, 0.95, '',
    transform=ax.transAxes,
                           ha='left', va='top', fontsize=12,

    bbox=dict(boxstyle="round,pad=0.4",
    facecolor="lightyellow", alpha=0.9))

# -------------------------------
# Zeitentwicklung
# -------------------------------
class WavepacketEvolution:
    def __init__(self, initial_psi):
        self.psi = initial_psi.copy()
        self.time = 0.0

    def step(self, dt):
        """Split-Step Fourier mit komplexem Potential (keine
    Normierung, da APL absorbiert)."""
        # Halber kinetischer Schritt
        psi_k = np.fft.fft(self.psi)
        psi_k *= exp_T_half
        self.psi = np.fft.ifft(psi_k)

        # Vollständiger Potentialschritt (Real-Teil:
    Phasenverschiebung, Imaginär-Teil: Absorption)
        self.psi *= exp_V

        # Halber kinetischer Schritt
        psi_k = np.fft.fft(self.psi)
        psi_k *= exp_T_half
        self.psi = np.fft.ifft(psi_k)

        # Keine Normierung: Die Norm sinkt durch die
    Absorption am Rand!
        self.time += dt

# Initialisiere die Evolution
evolution = WavepacketEvolution(psi)

def animate(frame):
    """Aktualisiert die Animation für jeden Frame."""
    for _ in range(substeps):
        evolution.step(dt)

```

```
    # Aktualisiere die rote Kurve
    prob_density = np.abs(evolution.psi)**2
    line_prob.set_ydata(prob_density)

    # TEXT basierend auf der Phase - mit Fokus auf dem Modell
    if evolution.time < 0.25:
        txt = (r"Phase 1: Ein freies Wellenpaket bewegt sich im ungestörten Vakuum." + "\n" +
               r"Die Geometrie ist noch homogen ($\gamma \approx \text{const}$).")
    elif evolution.time < 0.9:
        txt = (r"Phase 2: Das Paket erreicht die Domain Wall bei $r = r_c$." + "\n" +
               r"Die geometrische Deformation $\gamma(r)$ erzeugt hier ein" + "\n" +
               r"effektives Potential $V_{\mathrm{geom}}$, das aus der Raumkrümmung resultiert.")
    elif evolution.time < 1.2:
        txt = (r"Phase 3: Am Potentialtopf spaltet sich das Paket:" + "\n" +
               r"Ein Teil wird reflektiert, ein Teil dringt ein." + "\n" +
               r"Die Geometrie fungiert als Streuzentrum," + "\n" +
               r"rein durch Form, nicht durch Materie.")
    else:
        txt = (r"Phase 4: Der eingefangene Anteil oszilliert im Potentialtopf." + "\n" +
               r"Dies ist der lokalisierte Vakuumzustand, den das $\gamma$-Modell vorhersagt:" + "\n" +
               r"Geometrie erzeugt geschützte Quantenzustände.")

    explanation_text.set_text(txt)

    return line_prob, explanation_text

# ------------------------------
# Animation starten
# ------------------------------
print("Starte Animation...")
print("Beobachtung: Das Wellenpaket startet links, läuft nach rechts, wechselwirkt mit dem Potential bei r=2.5 und verliert Energie am rechten Rand (APL).")
```

```
anim = FuncAnimation(fig, animate, frames=frames,
    interval=1000/fps, blit=True)

# Zum Speichern auskommentieren:
anim.save('domain_wall_evolution_animation.gif',
    writer='pillow', fps=fps, dpi=120)

plt.show()
plt.close()
print(f" Stabile Animation mit dem Complex Absorbing
    Potential (APL) erstellt.")
```

Listing A.3: Visualisierung Domain Wall Localisation (Animation)

A.4 Dynamischer Casimir-Effekt (Animation), (Abschnitt. 3.5)

```
# dynamic_casimir_effect_gif_animation.py
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

# --------------------------------
# Physikalische Parameter
# --------------------------------
r_min, r_max = 1.0, 3.0
N = 1500
r = np.linspace(r_min, r_max, N)
dr = r[1] - r[0]

r_c = 2.0
w = 0.2
sigma0 = 2.0
delta_sigma = 1.5
omega = 7.0

def V_eff(r, sigma):
    d_gamma = sigma / w * (1.0 / np.cosh((r - r_c) / w))**2
    d2_gamma = -2 * sigma / (w**2) * np.tanh((r - r_c) / w)
    * (1.0 / np.cosh((r - r_c) / w))**2
    return -d2_gamma + 0.5 * d_gamma**2
```

```
24
25 # ------------------------------
26 # Anfangszustand: Grundzustand (statisch)
27 # ------------------------------
28 V0 = V_eff(r, sigma0)
29 psi = np.exp(-1.0 * (r - r_c)**2)  # Lokalisiert im Topf
30 psi = psi.astype(complex)
31 psi /= np.sqrt(np.trapezoid(np.abs(psi)**2, r))
32
33 # Berechne Anfangsenergie
34 dpsi_dr = np.gradient(psi, dr)
35 H_psi = -0.5 * np.gradient(dpsi_dr, dr) + V0 * psi
36 E0 = np.trapezoid(np.conj(psi) * H_psi, r).real
37
38 # ------------------------------
39 # ANIMATION
40 # ------------------------------
41 total_duration = 16.0
42 fps = 20
43 frames = int(total_duration * fps)
44 t_max = 4.0
45 substeps = 25
46 dt = t_max / (frames * substeps)
47
48 k = 2 * np.pi * np.fft.fftfreq(N, d=dr)
49
50 # ------------------------------
51 # Plot: ZWEI SUBPLOTS
52 # ------------------------------
53 fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))
54 plt.subplots_adjust(bottom=0.15, top=0.85, wspace=0.3)
55
56 fig.suptitle(r'Geometrisches Quantenvakuum', fontsize=14,
       fontweight='bold', y=0.96)
57 fig.text(0.5, 0.9, r'Visualisierung: Klaus H. Dieckmann,
       2025',
58          ha='center', fontsize=11, style='italic')
59
60 # Linker Plot: Potential
61 line_pot, = ax1.plot(r, V0 * 0.1, 'k-', linewidth=2,
       label=r'$0.1 \cdot V_{\mathrm{geom}}$')
62 ax1.axvline(x=r_c, color='gray', linestyle=':', alpha=0.8)
63 ax1.set_xlim(r_min, r_max)
64 ax1.set_ylim(-0.1, 1.2)
```

```
ax1.set_xlabel(r'$r$')
ax1.set_ylabel(r'Potential (skaliert)')
ax1.set_title('Oszillierende Domain Wall')
ax1.grid(True, linestyle='--', alpha=0.5)

# Rechter Plot: Energie über Zeit
times = []
energies = []
line_energy, = ax2.plot([], [], 'b-', linewidth=2.5)
ax2.set_xlim(0, t_max)
ax2.set_ylim(0, E0 * 3)
ax2.set_xlabel(r'Physikalische Zeit $t$')
ax2.set_ylabel(r'Gesamtenergie $E(t)$')
ax2.set_title('Teilchenerzeugung')
ax2.grid(True, linestyle='--', alpha=0.5)
ax2.axhline(y=E0, color='r', linestyle='--', alpha=0.7,
    label=r'$E_0$ (Anfang)')
ax2.legend()

# Dynamischer Text (unten)
explanation_text = fig.text(0.5, 0.02, '', ha='center',
    fontsize=11,

    bbox=dict(boxstyle="round,pad=0.4",
    facecolor="lightyellow", alpha=0.9))

# ------------------------------
# Zeitentwicklung
# ------------------------------
global_psi = psi.copy()
current_time = 0.0

def animate(frame):
    global global_psi, current_time, times, energies

    for _ in range(substeps):
        sigma_now = sigma0 + delta_sigma * np.sin(omega *
    current_time)
        V_now = V_eff(r, sigma_now)

        # Split-Step
        psi_k = np.fft.fft(global_psi)
        psi_k *= np.exp(-0.25j * k**2 * dt)
        global_psi = np.fft.ifft(psi_k)
```

```
        global_psi *= np.exp(-1j * V_now * dt)
        psi_k = np.fft.fft(global_psi)
        psi_k *= np.exp(-0.25j * k**2 * dt)
        global_psi = np.fft.ifft(psi_k)

        current_time += dt

    # Aktualisiere Potential-Plot
    sigma_now = sigma0 + delta_sigma * np.sin(omega *
    current_time)
    V_now = V_eff(r, sigma_now)
    line_pot.set_ydata(V_now * 0.1)

    # Berechne Gesamtenergie
    dpsi_dr = np.gradient(global_psi, dr)
    H_psi = -0.5 * np.gradient(dpsi_dr, dr) + V_now *
    global_psi
    E = np.trapezoid(np.conj(global_psi) * H_psi, r).real
    times.append(current_time)
    energies.append(E)

    # Aktualisiere Energie-Plot
    line_energy.set_data(times, energies)

    # Dynamischer Erklärtext - physikalisch korrekt
    if current_time < 0.6:
        txt = r"Phase 1: System im Vakuumgrundzustand. Keine
    Geometriemodulation. Energie ist konstant."
    elif current_time < 1.8:
        txt = (r"Phase 2: Die oszillierende Geometrie
    koppelt an Vakuummoden. "
            r"Ab dem ersten Schwingungszyklus" + "\n" +
            r"werden Photonenpaare erzeugt, sichtbar als
    kontinuierlicher Energieanstieg.")
    elif current_time < 3.0:
        txt = (r"Phase 3: Parametrische Resonanz verstärkt
    die Erzeugungsrate." + "\n" +
            r"Die Energie wächst exponentiell moduliert.")
    else:
        txt = (r"Phase 4: Kummulierte Teilchenzahl ist
    deutlich sichtbar. " + "\n" +
            r"Die Geometrie hat messbare Energie aus dem
    Vakuum extrahiert.")

```

```
    explanation_text.set_text(txt)
    return line_pot, line_energy, explanation_text

# ------------------------------
# Speichern
# ------------------------------
print("Erstelle klare Energie-basierte Animation...")
anim = FuncAnimation(fig, animate, frames=frames,
    interval=1000/fps, blit=False)
anim.save('dynamic_casimir_energy_animation.gif',
    writer='pillow', fps=fps, dpi=120)
plt.show()
plt.close()

print("✅ Klare Animation erstellt:
    'dynamic_casimir_energy_animation.gif'")
```

Listing A.4: Visualisierung Dynamischer Casimir-Effekt (Animation)

A.5 Entanglement Quench (Animation), (Abschnitt. 6)

```
# entanglement_quench_gif_animation.py
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

# ------------------------------
# Parameter
# ------------------------------
total_duration = 16.0
fps = 20
frames = int(total_duration * fps)
t_max = 4.0
quench_time = 0.8
gamma_max = 1.2

# ------------------------------
# Geometriefunktion MIT SYMMETRIELINIEN
# ------------------------------
def plot_cavity(ax, gamma):
    theta = np.linspace(0, 2*np.pi, 500)
```

```
    f_theta = np.cos(theta)**4 + np.sin(theta)**4
    r_vals = np.ones_like(theta)

    if gamma > 1e-6:
        for i, f in enumerate(f_theta):
            a = gamma * f
            disc = 1 + 4*a
            u = (-1 + np.sqrt(disc)) / (2*a)
            r_vals[i] = np.sqrt(max(u, 0.01))

    x = r_vals * np.cos(theta)
    y = r_vals * np.sin(theta)

    ax.clear()

    # Kavität
    ax.plot(x, y, 'b-', linewidth=2.5, label=r'Kavität:
$x^2+y^2+\gamma(x^4+y^4)=1$')

    # Symmetrielinien: x- und y-Achse „("Ecken)
    ax.axhline(0, color='gray', linestyle='-', alpha=0.4,
linewidth=1)
    ax.axvline(0, color='gray', linestyle='-', alpha=0.4,
linewidth=1)

    # Diagonalen (gestrichelt - dort ist die Form eingezogen)
    ax.plot([-1.2, 1.2], [-1.2, 1.2], color='gray',
linestyle='--', alpha=0.3, linewidth=0.8)
    ax.plot([-1.2, 1.2], [1.2, -1.2], color='gray',
linestyle='--', alpha=0.3, linewidth=0.8)

    ax.set_xlim(-1.3, 1.3)
    ax.set_ylim(-1.3, 1.3)
    ax.set_aspect('equal')
    ax.axis('off')

    # Titel mit γ
    ax.set_title(rf'$\gamma = {gamma:.2f}$', fontsize=12,
pad=10)

    # Statische Beschriftung (einmalig)
    if gamma == 0.0:
        ax.text(0, -1.15, r'Geometrie der
Quantenvakuum-Kavität',
```

```
                ha='center', va='top', fontsize=10,
    alpha=0.7)

# ------------------------------
# Entropie (monotone Relaxation)
# ------------------------------
time_points = np.linspace(0, t_max, frames)
S_ent = np.zeros_like(time_points)

S_initial = 0.50
S_final = 0.50 - 0.12 * gamma_max**2

for i, t in enumerate(time_points):
    if t < quench_time:
        S_ent[i] = S_initial
    else:
        dt = t - quench_time
        S_ent[i] = S_final + (S_initial - S_final) *
    np.exp(-dt / 1.8)

# ------------------------------
# Plot
# ------------------------------
fig, (ax_top, ax_bottom) = plt.subplots(2, 1, figsize=(8,
    7), gridspec_kw={'height_ratios': [1.2, 1]})
plt.subplots_adjust(bottom=0.12, top=0.88, hspace=0.3)

fig.suptitle(r'Geometrisches Quantenvakuum: Entanglement
    Quench', fontsize=14,  fontweight='bold', y=0.96)
fig.text(0.5, 0.46, r'Visualisierung: Klaus H. Dieckmann,
    2025',
         ha='center', fontsize=12, style='italic')

plot_cavity(ax_top, 0.0)

line_ent, = ax_bottom.plot([], [], 'm-', linewidth=2.5)
ax_bottom.axvline(x=quench_time, color='red',
    linestyle='--', alpha=0.7, label=r'Quench bei $t_0$')
ax_bottom.set_xlim(0, t_max)
ax_bottom.set_ylim(0.25, 0.52)
ax_bottom.set_xlabel(r'Physikalische Zeit $t$')
ax_bottom.set_ylabel(r'Verschränkungs-Entropie
    $S_{\mathrm{ent}}$')
ax_bottom.legend(loc='lower right')
```

```
ax_bottom.grid(True, linestyle='--', alpha=0.5)

explanation_text = fig.text(0.5, 0.02, '', ha='center',
    fontsize=11,

    bbox=dict(boxstyle="round,pad=0.4",
    facecolor="lightyellow", alpha=0.9))

# ------------------------------
# Animation
# ------------------------------
def animate(frame):
    t = time_points[frame]
    gamma_now = 0.0 if t < quench_time else gamma_max
    plot_cavity(ax_top, gamma_now)
    line_ent.set_data(time_points[:frame+1], S_ent[:frame+1])

    if t < quench_time:
        txt = (r"Phase 1: Rotationssymmetrische Kavität
    ($\gamma=0$). " + "\n" +
               r"Maximale Verschränkung.")
    elif t < quench_time + 0.3:
        txt = (r"Phase 2: Quanten-Quench! $\gamma \to
    \gamma_{\mathrm{max}}$. " + "\n" +
               r"Kavität wird anisotrop ('eckig').
    Verschränkung bricht ein.")
    elif t < quench_time + 2.5:
        txt = (r"Phase 3: Relaxation ins neue Gleichgewicht.
    Die anisotrope Geometrie " + "\n" +
               r"reduziert die Quantenkorrelationen
    dauerhaft.")
    else:
        txt = (r"Phase 4: Neues Gleichgewicht. Geometrie als
    " + "\n" +
               r"Steuerparameter für Quanteninformation.")

    explanation_text.set_text(txt)
    return line_ent, explanation_text

# ------------------------------
# Speichern
# ------------------------------
print("Erstelle kontextreiche Animation...")
```

```
anim = FuncAnimation(fig, animate, frames=frames,
    interval=1000/fps, blit=False)
anim.save('entanglement_quench_context_animation.gif',
    writer='pillow', fps=fps, dpi=120)
plt.show()
plt.close()

print("□ GIF-Animation erstellt!")
```

Listing A.5: Visualisierung Entanglement Quench (Animation)

A.6 Anisotrope Dispersion (Animation), (Abschnitt. 6.3)

```
# anisotropic_propagation_gif_animation.py
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

# ------------------------------
# Simulationsparameter
# ------------------------------
L = 8.0          # Größere Box (offenes System)
N = 384          # Höhere Auflösung
x = np.linspace(-L, L, N)
y = np.linspace(-L, L, N)
X, Y = np.meshgrid(x, y)

# ------------------------------
# Anfangszustand: Isotropes Gauß-Paket mit Impuls
# ------------------------------
x0, y0 = 0.0, 0.0
sigma0 = 0.4     # Breiteres Paket
k0 = 0.0         # Kein Impuls - isotrope Ausbreitung
psi = np.exp(1j * k0 * (X - x0)) * np.exp(-0.5 * ((X -
    x0)**2 + (Y - y0)**2) / sigma0**2)
psi = psi.astype(complex)

# Normierung
norm = np.sqrt(np.sum(np.abs(psi)**2) * (x[1]-x[0]) *
    (y[1]-y[0]))
psi /= norm
```

```
27
28 # ------------------------------
29 # ANIMATION
30 # ------------------------------
31 total_duration = 14.0
32 fps = 20
33 frames = int(total_duration * fps)
34 t_max = 3.0
35 substeps = 10
36 dt = t_max / (frames * substeps)
37
38 # ------------------------------
39 # Numerik: Anisotrope Dispersion
40 # ------------------------------
41 dx = x[1] - x[0]
42 kx = 2 * np.pi * np.fft.fftfreq(N, d=dx)
43 ky = 2 * np.pi * np.fft.fftfreq(N, d=dx)
44 KX, KY = np.meshgrid(kx, ky)
45
46 # Isotroper Term: k²
47 K2 = KX**2 + KY**2
48
49 # Anisotroper Term: ⁴k_x + ⁴k_y
50 K4 = KX**4 + KY**4
51
52 alpha = 0.08  # Stärke der Anisotropie
53
54 # Zeitentwicklungsoperator (Fourier-Raum)
55 exp_op = np.exp(-0.5j * (K2 + alpha * K4) * dt)
56
57 # ------------------------------
58 # Plot
59 # ------------------------------
60 fig, ax = plt.subplots(figsize=(8, 7))
61 #plt.subplots_adjust(bottom=0.12, top=0.88)
62 plt.subplots_adjust(bottom=0.13, top=0.8)
63
64 fig.suptitle(r'Geometrisches Quantenvakuum: Anisotrope
       Dispersion', fontsize=14, fontweight='bold', y=0.96)
65 fig.text(0.5, 0.89, r'Visualisierung: Klaus H. Dieckmann,
       2025',
66          ha='center', fontsize=12, style='italic')
67
68 # Heatmap
```

```
im = ax.imshow(np.abs(psi)**2, extent=[-L, L, -L, L],
    origin='lower',
                cmap='magma', vmin=0, vmax=0.02)
ax.set_xlabel(r'$x$')
ax.set_ylabel(r'$y$')
ax.set_title(r'$|\psi(x,y,t)|^2$ mit anisotroper
    Dispersion', fontsize=11)

# Symmetrielinien
ax.axhline(0, color='white', linestyle='-', alpha=0.6,
    linewidth=0.9)
ax.axvline(0, color='white', linestyle='-', alpha=0.6,
    linewidth=0.9)
ax.plot([-L, L], [-L, L], color='white', linestyle='--',
    alpha=0.4, linewidth=0.7)
ax.plot([-L, L], [L, -L], color='white', linestyle='--',
    alpha=0.4, linewidth=0.7)

cbar = plt.colorbar(im, ax=ax, fraction=0.046, pad=0.04)
cbar.set_label(r'$|\psi|^2$', rotation=0, labelpad=15)

explanation_text = fig.text(0.5, 0.02, '', ha='center',
    fontsize=11,

    bbox=dict(boxstyle="round,pad=0.4",
    facecolor="lightyellow", alpha=0.9))

# ------------------------------
# Zeitentwicklung
# ------------------------------
global_psi = psi.copy()
current_time = 0.0

def animate(frame):
    global global_psi, current_time

    for _ in range(substeps):
        # Fourier-Transformation
        psi_k = np.fft.fft2(global_psi)
        # Anisotrope Zeitentwicklung
        psi_k *= exp_op
        global_psi = np.fft.ifft2(psi_k)
        current_time += dt

```

```
    # Update Heatmap
    prob = np.abs(global_psi)**2
    im.set_array(prob)

    # Erklärtext
    if current_time < 0.5:
        txt = r"Start: Isotropes Wellenpaket in freiem Raum."
    elif current_time < 1.2:
        txt = (r"Ausbreitung beginnt: Die anisotrope Dispersion beschleunigt die " + "\n" +
               r"Ausbreitung entlang der $x$- und $y$-Achsen.")
    elif current_time < 2.2:
        txt = (r"Deutliche Richtungsabhängigkeit: Das Paket wird 'kreuzförmig'.  " + "\n" +
               r"Diagonalen hinken hinterher, wie bei Doppelbrechung.")
    else:
        txt = (r"Effektive Metrik des Vakuums: Die Geometrie (hier: Dispersion)  " + "\n" +
               r"steuert die Quantendynamik richtungsabhängig.")

    explanation_text.set_text(txt)
    return im, explanation_text

# ------------------------------
# GIF speichern
# ------------------------------
print("Erstelle Animation mit anisotroper Dispersion...")
anim = FuncAnimation(fig, animate, frames=frames, interval=1000/fps, blit=False)
anim.save('anisotropic_dispersion_animation.gif', writer='pillow', fps=fps, dpi=120)
plt.show()
plt.close()

print("✅ GIF 'anisotropic_dispersion_animation.gif' wurde erstellt!")
print(f"   Anisotropie-Parameter α = {alpha}")
```

Listing A.6: Visualisierung Anisotrope Dispersion (Animation)

A.7 Quantencomputer Qubits Beweis, (Abschnitt. 5.5.1)

```python
# quantencomputer_qubit_beweis.py
import numpy as np
import matplotlib.pyplot as plt

# Physikalische Parameter (in willkürlichen, aber konsistenten Einheiten)
V0 = 100.0        # Potentialtiefe
w = 0.5           # Breite der Domain Wall
L = 10.0          # Simulationsbox-Halblänge (Gesamtlänge = 2*L)
N = 2000          # Anzahl der Gitterpunkte

# Gitter erstellen
x = np.linspace(-L, L, N)
dx = x[1] - x[0]

# Potential definieren: Pöschl-Teller-artig
V = -V0 / np.cosh((x) / w)**2

# Hamilton-Matrix aufstellen (dichte Matrix)
# Kinetische Energie: -d^2/dx^2 -> Finite-Differenzen mit 1/dx^2 Skalierung
H = np.zeros((N, N))
for i in range(N):
    H[i, i] = 2.0 / dx**2 + V[i]  # Diagonalelement
    if i > 0:
        H[i, i-1] = -1.0 / dx**2   # Untere Nebendiagonale
    if i < N-1:
        H[i, i+1] = -1.0 / dx**2   # Obere Nebendiagonale

# Nur die niedrigsten Eigenzustände berechnen (die relevanten gebundenen Zustände)
print("Berechne Eigenzustände...")
# Da wir die volle Matrix haben, verwenden wir eigh. Es berechnet alle, aber wir nehmen nur die ersten.
eigvals_all, eigvecs_all = np.linalg.eigh(H)
# Die Eigenwerte sind aufsteigend sortiert. Wir nehmen die ersten beiden.
eigvals = eigvals_all[:6]
eigvecs = eigvecs_all[:, :6]
```

```
35
36 # Normierung der Wellenfunktionen (kontinuierliche Norm:
       □ψ||² dx = 1)
37 psi0 = eigvecs[:, 0]
38 psi1 = eigvecs[:, 1]
39 norm0 = np.sqrt(np.trapezoid(np.abs(psi0)**2, x))
40 norm1 = np.sqrt(np.trapezoid(np.abs(psi1)**2, x))
41 psi0 /= norm0
42 psi1 /= norm1
43
44 # Physikalische Größen berechnen
45 E0 = eigvals[0]
46 E1 = eigvals[1]
47 omega0 = E1 - E0
48
49 # Dipolmatrixelement <1|x|0>
50 d01 = np.trapezoid(psi1 * x * psi0, x)
51
52 print("\n=== ERGEBNISSE ===")
53 print(f"E0 = {E0:.4f}")
54 print(f"E1 = {E1:.4f}")
55 print(f"ω0 = {omega0:.4f} (im Mikrowellenbereich!)")
56 print(f"<1|x|0> = {d01:.4f}")
57
58 # Kopplung J berechnen für zwei Domains
59 def calculate_coupling(d):
60     """Berechnet die Kopplungsstärke J für zwei Domains im
       Abstand d."""
61     # Zwei Potentiale bei -d/2 und +d/2
62     V_double = -V0 / np.cosh((x + d/2) / w)**2 - V0 /
       np.cosh((x - d/2) / w)**2
63
64     H_double = np.zeros((N, N))
65     for i in range(N):
66         H_double[i, i] = 2.0 / dx**2 + V_double[i]
67         if i > 0:
68             H_double[i, i-1] = -1.0 / dx**2
69         if i < N-1:
70             H_double[i, i+1] = -1.0 / dx**2
71
72     # Die beiden niedrigsten Zustände des Doppelpotentials
       berechnen
73     eigvals_double, eigvecs_double = np.linalg.eigh(H_double)
74     E_sym = eigvals_double[0]
```

```
    E_asym = eigvals_double[1]
    # Die Kopplungsstärke ist die halbe Aufspaltung
    J_approx = (E_asym - E_sym) / 2.0
    return J_approx

# Kopplung für einen repräsentativen Abstand berechnen
d_test = 2.0  # Abstand in Einheiten der Simulationsbox
J_val = calculate_coupling(d_test)
print(f"J(d={d_test}) = {J_val:.6f} (exponentiell klein!)")

# Plots erstellen
plt.figure(figsize=(12, 5))

# Plot 1: Potential und Wellenfunktionen
plt.subplot(1, 2, 1)
plt.plot(x, V, 'k-', label='Potential $V(x)$', linewidth=2)
plt.plot(x, psi0**2 * 50 + E0, 'b--',
    label='$|\\psi_0(x)|^2$ (skaliert)', alpha=0.8)
plt.plot(x, psi1**2 * 50 + E1, 'r--',
    label='$|\\psi_1(x)|^2$ (skaliert)', alpha=0.8)
plt.axhline(E0, color='b', linestyle=':', alpha=0.5)
plt.axhline(E1, color='r', linestyle=':', alpha=0.5)
plt.xlabel('$x$')
plt.ylabel('Energie / Amplitude')
plt.title('Lokalisierte Zustände an der Domain Wall')
plt.legend()
plt.grid(True, alpha=0.3)

# Plot 2: Kopplungsstärke J als Funktion des Abstands
d_vals = np.linspace(0.5, 4.0, 10) # Weniger Punkte für
    schnellere Berechnung
J_vals = [calculate_coupling(d) for d in d_vals]

plt.subplot(1, 2, 2)
plt.semilogy(d_vals, J_vals, 'bo-', label='$J(d)$')
plt.xlabel('Abstand $d$')
plt.ylabel('Kopplungsstärke $J$')
plt.title('Exponentieller Abfall der Kopplung')
plt.legend()
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('quantencomputer_qubit_results.png', dpi=150) #
    Speichern kann in manchen Umgebungen fehlschlagen
```

```
plt.show()

print("\nDie Kopplung J ~ exp(-d/w) mit ≈w0.5 ist der
    Schlüssel zur Skalierbarkeit.")
```

Listing A.7: Visualisierung Quantencomputer Qubits Beweis

A.8 Quantencomputer: Qubit-Dynamik (Animation), (Abschnitt. 5)

```
# quantencomputer_qubit_gif_animation.py
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation, PillowWriter

# ------------------------------
# 1. Physikalische Parameter und Setup
# ------------------------------
V0 = 100.0        # Potentialtiefe
w = 0.5           # Breite der Domain Wall
L = 10.0          # Simulationsbox-Halblänge
N = 2000          # Anzahl der Gitterpunkte
x = np.linspace(-L, L, N)
dx = x[1] - x[0]

# Potential definieren
V = -V0 / np.cosh((x) / w)**2

# Hamilton-Matrix aufstellen
H = np.zeros((N, N))
for i in range(N):
    H[i, i] = 2.0 / dx**2 + V[i]
    if i > 0:
        H[i, i-1] = -1.0 / dx**2
    if i < N-1:
        H[i, i+1] = -1.0 / dx**2

# Eigenzustände berechnen
eigvals_all, eigvecs_all = np.linalg.eigh(H)
psi0 = eigvecs_all[:, 0]
psi1 = eigvecs_all[:, 1]

```

```
# Normierung
norm0 = np.sqrt(np.trapezoid(np.abs(psi0)**2, x))
norm1 = np.sqrt(np.trapezoid(np.abs(psi1)**2, x))
psi0 /= norm0
psi1 /= norm1

E0 = eigvals_all[0]
E1 = eigvals_all[1]
omega0 = E1 - E0

# -----------------------------
# 2. Zeitliche Entwicklung für die Animation (PHASENGESTEUERT)
# -----------------------------
# Neue Strategie: Jede Phase dauert genau 5 Sekunden.
phase_duration_sec = 5
num_phases = 5
total_duration_sec = phase_duration_sec * num_phases
fps = 20  # Etwas reduziert für kleinere Dateigröße
total_frames = total_duration_sec * fps

# Die physikalische Periode der Oszillation bleibt unverändert
T_period = 2 * np.pi / omega0

# Anfangszustand: |psi(0)> = (|0> + |1>) / sqrt(2)
def psi_t(t):
    return (psi0 * np.exp(-1j * E0 * t) + psi1 * np.exp(-1j * E1 * t)) / np.sqrt(2)

# Listen für die Animation
prob_densities = []
phase_labels = []  # Speichert, in welcher Phase wir uns pro Frame befinden

print("Berechne Phasen für die Animation...")
for frame in range(total_frames):
    # 1. Bestimme die Animationszeit (0 bis total_duration_sec)
    t_anim = frame / fps

    # 2. Bestimme die aktuelle Phase (0, 1, 2, 3, 4)
    current_phase = int(t_anim // phase_duration_sec)
```

```
    current_phase = min(current_phase, num_phases - 1) # Sicherheitscheck
    phase_labels.append(current_phase)

    # 3. Berechne die zugehörige SIMULATIONSZEIT (t_sim)
    #    Jede Phase deckt 1/4 der physikalischen Periode ab (0->T/4, T/4->T/2, etc.)
    if current_phase == 0:
        # Phase 0: Initialisierung (zeige nur den Anfangszustand)
        t_sim = 0.0
    elif current_phase == 4:
        # Phase 4: Messvorbereitung (zeige den Zustand bei T/2)
        t_sim = T_period / 2
    else:
        # Phase 1-3: Zeige die Oszillation zwischen den Schlüsselpunkten
        # Phase 1: 0 -> T/4
        # Phase 2: T/4 -> T/2
        # Phase 3: T/2 -> 3T/4
        phase_start_time = (current_phase - 1) * (T_period / 4)
        phase_end_time = current_phase * (T_period / 4)
        # Wo sind wir innerhalb der aktuellen Animationsphase?
        phase_progress = (t_anim % phase_duration_sec) / phase_duration_sec
        t_sim = phase_start_time + phase_progress * (phase_end_time - phase_start_time)

    # 4. Berechne den Zustand und speichere ihn
    psi = psi_t(t_sim)
    prob_densities.append(np.abs(psi)**2)

# ----------------------------
# 3. Animation mit dynamischem Text
# ----------------------------
fig, ax = plt.subplots(figsize=(10, 6))
plt.subplots_adjust(bottom=0.25, top=0.85)
ax.set_xlim(-L, L)
ax.set_ylim(0, np.max(prob_densities) * 0.85)
ax.set_xlabel('$x$', fontsize=14)
```

```
ax.set_ylabel('Wahrscheinlichkeitsdichte $|\psi(x,t)|^2$',
    fontsize=14)
ax.grid(True, alpha=0.3)

# Statische Elemente
potential_line, = ax.plot(x, V/np.max(np.abs(V)) *
    np.max(prob_densities)*0.8, 'k-', linewidth=2,
    label='Domain-Wall-Potential')
density_line, = ax.plot([], [], 'b-', linewidth=2.5,
    label='Qubit-Zustand')
ax.legend(loc='upper right')

# Titel und Untertitel
fig.suptitle('Geometrisches Quantenvakuum: Qubit-Dynamik',
    fontsize=14, fontweight='bold')
fig.text(0.5, 0.92, 'Visualisierung: Klaus H. Dieckmann,
    2025', ha='center', fontsize=12)

# Dynamischer Erklärtext
explanation_text = fig.text(0.5, 0.08, '', ha='center',
    fontsize=11,fontweight='bold', wrap=True,

    bbox=dict(boxstyle="round,pad=0.3",
    facecolor="lightgray", alpha=0.7))

def animate(frame):
    density_line.set_data(x, prob_densities[frame])

    # Verwende die vorab berechnete Phase
    current_phase = phase_labels[frame]

    # Stark verkürzte, aber nun ausreichend lange Erklärtexte
    if current_phase == 0:
        text = "Phase 1: Initialisierung.\nQubit im Zustand
    |+>."
    elif current_phase == 1:
        text = "Phase 2: Kohärente Oszillation.\nEntspricht
    einem Quantengatter."
    elif current_phase == 2:
        text = "Phase 3: Maximale
    Asymmetrie.\nQuanteninformation für Quantenalgorithmen
    ist kodiert."
    elif current_phase == 3:
```

```
        text = "Phase 4: Zyklische Rückkehr.\nGrundlage für Quantenoperationen in einem Quantencomputer"
    else: # current_phase == 4
        text = "Phase 5: Messvorbereitung.\nKollaps zu 0 oder 1 (Ausgabe)."

    explanation_text.set_text(text)
    return density_line, explanation_text

# ------------------------------
# 4. GIF erstellen
# ------------------------------
print("Erstelle Animation... (dies kann einige Sekunden dauern)")
anim = FuncAnimation(fig, animate, frames=total_frames, interval=1000/fps, blit=False, repeat=True)

# GIF speichern
writer = PillowWriter(fps=fps)
anim.save('quantencomputer_qubit_animation.gif', writer=writer, dpi=150)
print("Animation 'quantencomputer_qubit_animation.gif' wurde erfolgreich erstellt.")
plt.show()
plt.close()
```

Listing A.8: Visualisierung Quantencomputer: Qubit-Dynamik (Animation)

A.9 Casimir Torque Rotation (Animation), (Abschn. 3.5)

```
# casimir_torque_rotation_animation.py
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

# ------------------------------
# Parameter
# ------------------------------
gamma = 1.2  # Deformationsstärke
theta_max = 90  # Grad
```

```
frames = 180
fps = 20

# Winkel in Grad und Bogenmaß
theta_deg_full = np.linspace(0, theta_max, 500)
theta_rad_full = np.deg2rad(theta_deg_full)

# Drehmoment: τθ() □ sin³θ(2) cosθ(2)
torque = np.sin(2 * theta_rad_full)**3 * np.cos(2 *
    theta_rad_full)

# ------------------------------
# Hilfsfunktion: Kavität zeichnen
# ------------------------------
def cavity_shape(gamma, rotation_angle_deg=0):
    """Gibt x, y für eine y-deformierte Kavität zurück,
    optional rotiert."""
    theta = np.linspace(0, 2 * np.pi, 400)
    f = np.cos(theta)**4 + np.sin(theta)**4
    r = np.ones_like(theta)
    for i, f_val in enumerate(f):
        if gamma > 1e-6:
            a = gamma * f_val
            disc = 1 + 4 * a
            u = (-1 + np.sqrt(disc)) / (2 * a)
            r[i] = np.sqrt(max(u, 0.01))
    x = r * np.cos(theta)
    y = r * np.sin(theta)

    # Rotation anwenden
    if rotation_angle_deg != 0:
        angle = np.deg2rad(rotation_angle_deg)
        x_rot = x * np.cos(angle) - y * np.sin(angle)
        y_rot = x * np.sin(angle) + y * np.cos(angle)
        return x_rot, y_rot
    return x, y

# ------------------------------
# Plot vorbereiten
# ------------------------------
fig, (ax_top, ax_bottom) = plt.subplots(2, 1, figsize=(8,
    9), gridspec_kw={'height_ratios': [1.2, 1]})
plt.subplots_adjust(hspace=0.3)

```

```
# Titel und Untertitel
fig.suptitle(r'Geometrisches Quantenvakuum', fontsize=14,
    fontweight='bold', y=0.96)
fig.text(0.5, 0.5, r'Relativ drehende ₄C-symmetrische
    Kavitäten',
         ha='center', fontsize=12)
fig.text(0.5, 0.89, r'Visualisierung: Klaus H. Dieckmann,
    2025',
         ha='center', fontsize=11, style='italic')

# Oben: Kavitäten
ax_top.set_xlim(-1.4, 1.4)
ax_top.set_ylim(-1.4, 1.4)
ax_top.set_aspect('equal')
ax_top.axis('off')
#ax_top.set_title('Relativ drehende ₄C-symmetrische
    Kavitäten', fontsize=12)

# Unten: Drehmoment
ax_bottom.plot(theta_deg_full, torque, 'b-', linewidth=2,
    label=r'$\tau(\theta) \propto
    \sin^3(2\theta)\cos(2\theta)$')
marker_line, = ax_bottom.plot([], [], 'ro', markersize=8,
    label='Aktueller Zustand')
ax_bottom.set_xlim(0, theta_max)
ax_bottom.set_ylim(-0.7, 0.7)
ax_bottom.set_xlabel(r'Relativer Drehwinkel $\theta$ [Grad]')
ax_bottom.set_ylabel(r'Casimir-Drehmoment $\tau$ [arb.
    Einh.]')
ax_bottom.grid(True, linestyle='--', alpha=0.6)
ax_bottom.legend(loc='upper right')

# Textfeld
text_box = ax_bottom.text(0.02, 0.95, '',
    transform=ax_bottom.transAxes,
                          fontsize=11,
    verticalalignment='top',

    bbox=dict(boxstyle="round,pad=0.4",
    facecolor="lightyellow", alpha=0.9))

# ------------------------------
# Initialisierung
# ------------------------------
```

```
outer_x, outer_y = cavity_shape(gamma, 0)
inner_x, inner_y = cavity_shape(gamma, 0)

outer_line, = ax_top.plot(outer_x, outer_y, 'b-',
    linewidth=2.5, label='Äußere Kavität')
inner_line, = ax_top.plot(inner_x, inner_y, 'r--',
    linewidth=2, label='Innere Kavität (rotiert)')
ax_top.legend(loc='upper right')

# --------------------------------
# Animationsfunktion
# --------------------------------
def animate(frame):
    # Aktueller Winkel
    theta_deg = frame * theta_max / frames
    theta_rad = np.deg2rad(theta_deg)

    # Kavitäten aktualisieren
    _, outer_y = cavity_shape(gamma, 0)  # Äußere bleibt fest
    inner_x, inner_y = cavity_shape(gamma, theta_deg)
    outer_line.set_data(outer_x, outer_y)
    inner_line.set_data(inner_x, inner_y)

    # Drehmoment-Marker
    tau_val = np.sin(2 * theta_rad)**3 * np.cos(2 *
    theta_rad)
    marker_line.set_data([theta_deg], [tau_val])

    # Text aktualisieren
    is_max = abs(theta_deg - 30) < 1.0 or abs(theta_deg -
    60) < 1.0
    status = " → Drehmoment maximal bei 30°!" if is_max else
    ""
    text_box.set_text(f'Relativer Drehwinkel θ =
    {theta_deg:.1f}°{status}')

    return outer_line, inner_line, marker_line, text_box

# --------------------------------
# Animation starten und speichern
# --------------------------------
print("Erstelle Animation
    'casimir_torque_rotation_animation.gif'...")
```

```
anim = FuncAnimation(fig, animate, frames=frames,
    interval=1000/fps, blit=False, repeat=True)

# GIF speichern
anim.save('casimir_torque_rotation_animation.gif',
    writer='pillow', fps=fps, dpi=120)
plt.show()
plt.close()
print("□ Animation gespeichert als
    'casimir_torque_rotation_animation.gif'")
```

Listing A.9: Casimir Torque Rotation (Animation)

A.10 Eigenwert-Splitting (Animation), (Abschn. 4.2)

```
# eigenvalue_splitting_animation.py
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

# ------------------------------
# Parameter
# ------------------------------
gamma_max = 2.0
frames = 300
fps = 20

# Simulierte Eigenwerte (basierend auf Bessel-Entartung im
    Kreis)
k0 = np.array([2.4048, 3.8317, 3.8317, 5.1356, 5.1356])  #
    Entartete Paare bei y=0
gamma_vals = np.linspace(0, gamma_max, frames)
eigenvalues = np.zeros((frames, 5))
C = 0.18  # aus der Arbeit

for i, g in enumerate(gamma_vals):
    if g == 0:
        eigenvalues[i] = k0
    else:
        split = 0.15 * g
        eigenvalues[i] = np.array([
            k0[0],
            k0[1] - split,
```

```
            k0[2] + split,
            k0[3] - split,
            k0[4] + split
        ])

# Kumulierte Casimir-Energie (relativ)
EC = -C * gamma_vals**2

# ------------------------------
# Kavität zeichnen
# ------------------------------
def cavity_shape(gamma, num_points=400):
    theta = np.linspace(0, 2*np.pi, num_points)
    if gamma == 0:
        r = np.ones_like(theta)
    else:
        f = np.cos(theta)**4 + np.sin(theta)**4
        r = np.zeros_like(theta)
        for i, f_val in enumerate(f):
            a = gamma * f_val
            disc = 1 + 4*a
            u = (-1 + np.sqrt(disc)) / (2*a) if a != 0 else
    1.0
            r[i] = np.sqrt(max(u, 0.01))
    x = r * np.cos(theta)
    y = r * np.sin(theta)
    return x, y

# ------------------------------
# Plot vorbereiten
# ------------------------------
fig = plt.figure(figsize=(10, 6))
fig.suptitle(r'Geometrisches Quantenvakuum', fontsize=14,
    fontweight='bold', y=0.96)
fig.text(0.3, 0.89, r'Vergleich: Isotrop vs. Anisotrop',
         ha='center', fontsize=12)
fig.text(0.3, 0.85, r'Modenspektrum & Casimir-Energie',
         ha='center', fontsize=12)
fig.text(0.3, 0.2, r'Visualisierung: Klaus H. Dieckmann,
    2025',
         ha='center', fontsize=12, style='italic')

# Subplots
ax_left = plt.subplot2grid((2, 2), (0, 0), rowspan=2)
```

```
ax_right_top = plt.subplot2grid((2, 2), (0, 1))
ax_right_bottom = plt.subplot2grid((2, 2), (1, 1))

# Links: Kavität
ax_left.set_xlim(-1.4, 1.4)
ax_left.set_ylim(-1.4, 1.4)
ax_left.set_aspect('equal')
ax_left.axis('off')
cavity_line, = ax_left.plot([], [], 'b-', linewidth=2.5)
ax_left.set_title('Kavität: γ = 0 → 2.0', fontsize=11)

# Rechts oben: Eigenwerte
bars = ax_right_top.bar(range(1, 6), eigenvalues[0],
    color='steelblue')
ax_right_top.set_ylim(2.0, 5.8)
ax_right_top.set_ylabel(r'Eigenwerte $k_n(\gamma)$')
ax_right_top.set_xticks([1, 2, 3, 4, 5])
ax_right_top.set_title('Modenspektrum', fontsize=11)
ax_right_top.grid(True, linestyle='--', alpha=0.5)

# Rechts unten: Casimir-Energie
ec_line, = ax_right_bottom.plot([], [], 'r-', linewidth=2)
ax_right_bottom.set_xlim(0, gamma_max)
ax_right_bottom.set_ylim(EC.min() * 1.1, 0)
ax_right_bottom.set_xlabel(r'Deformationsparameter $\gamma$')
ax_right_bottom.set_ylabel(r'$E_C(\gamma)$ [rel.]')
ax_right_bottom.grid(True, linestyle='--', alpha=0.5)
ax_right_bottom.set_title('Casimir-Energie', fontsize=11)

# Dynamischer Text
explanation_text = fig.text(0.02, 0.02, '', fontsize=11,

    bbox=dict(boxstyle="round,pad=0.4",
    facecolor="lightyellow", alpha=0.9))

# ------------------------------
# Animationsfunktion
# ------------------------------
def animate(frame):
    gamma = gamma_vals[frame]

    # Kavität aktualisieren
    x, y = cavity_shape(gamma)
    cavity_line.set_data(x, y)
```

```
109
110     # Balkendiagramm aktualisieren
111     for bar, val in zip(bars, eigenvalues[frame]):
112         bar.set_height(val)
113
114     # Casimir-Energie aktualisieren
115     ec_line.set_data(gamma_vals[:frame+1], EC[:frame+1])
116
117     # Erklärtext
118     if gamma < 0.1:
119         txt = (r"Phase 1: Isotrope Kavität ($\gamma = 0$)."
    + "\n" +
120                r"Rotationssymmetrie ⇒ entartete Moden
    (z .B. $k_2 = k_3$).")
121     elif gamma < 1.0:
122         txt = (r"Phase 2: Anisotrope Deformation ($\gamma >
    0$)." + "\n" +
123                r"Symmetriebrechung hebt Entartung auf -
    Moden spalten sich auf.")
124     else:
125         txt = (r"Phase 3: Stark anisotrope Kavität." + "\n" +
126                r"Vollständige Aufspaltung des Spektrums.
    Casimir-Energie sinkt quadratisch.")
127
128     explanation_text.set_text(txt)
129
130     return cavity_line, ec_line, explanation_text, *bars
131
132 # ------------------------------
133 # Animation starten und speichern
134 # ------------------------------
135 print("Erstelle Animation
    'eigenvalue_splitting_animation.gif'...")
136 anim = FuncAnimation(fig, animate, frames=frames,
    interval=1000/fps, blit=False, repeat=True)
137
138 anim.save('eigenvalue_splitting_animation.gif',
    writer='pillow', fps=fps, dpi=120)
139 plt.show()
140 plt.close()
141 print("✅ Animation gespeichert als
    'eigenvalue_splitting_animation.gif'")
```

Listing A.10: Eigenwert-Splitting (Animation)

Hinweis zur Nutzung von KI

Die Ideen und Konzepte dieser Arbeit stammen von mir. Künstliche Intelligenz wurde unterstützend für die Textformulierung und Gleichungsformatierung eingesetzt. Die inhaltliche Verantwortung liegt bei mir. [1]

Stand: 8. Oktober 2025
TimeStamp: https://freetsa.org/index_de.php

[1] ORCID: https://orcid.org/0009-0002-6090-3757

Literatur

[1] C. Barceló, S. Liberati, and M. Visser, Analogue Gravity, *Living Rev. Relativ.* **14**, 3 (2011).

[2] Bimonte, G., Emig, T., Kardar, M., *Casimir interactions between anisotropic materials: Exact results for planar systems*, Phys. Rev. A, Vol. 95, 062514 (2017).

[3] L. Bombelli, R. K. Koul, J. Lee, and R. D. Sorkin, Quantum source of entropy for black holes, *Phys. Rev. D* **34**, 373–383 (1986).

[4] P. Calabrese and J. Cardy, Entanglement entropy and quantum field theory, *J. Stat. Mech.* P06002 (2004).

[5] C. G. Callan and F. Wilczek, On geometric entropy, *Phys. Lett. B* **333**, 55–61 (1994).

[6] Casimir, H. B. G. *On the attraction between two perfectly conducting plates*, Proc. Kon. Ned. Akad. Wetensch. **51**, 793–795 (1948).

[7] Ford, L. H. and Roman, T. A., *Averaged energy conditions and quantum inequalities*, Phys. Rev. D, Vol. 51, 4277–4286 (1995).

[8] R. Jackiw and C. Rebbi, Solitons with fermion number 1/2, *Phys. Rev. D* **13**, 3398–3409 (1976).

[9] A. Y. Kitaev, *Annals of Physics* **303**, 2 (2003).

[10] S. K. Lamoreaux, Demonstration of the Casimir Force in the 0.6 to 6 μm Range, *Phys. Rev. Lett.* **78**, 5–8 (1997).

[11] J. Maldacena and L. Susskind, Cool horizons for entangled black holes, *Fortsch. Phys.* **61**, 781–811 (2013).

[12] M. Minkov et al., *Optica* **3**, 124 (2016).

[13] Moore, G. T., *Quantum theory of the electromagnetic field in a variable-length one-dimensional cavity*, J. Math. Phys. **11**, 2679–2691 (1970).

[14] J. N. Munday, F. Capasso, and V. A. Parsegian, Measurement of the Casimir torque, *Nature* **564**, 386–389 (2018).

[15] M. Notomi et al., *Nature Photonics* **4**, 745 (2010).

[16] G. Vidal, J. I. Latorre, E. Rico, and A. Kitaev, Entanglement in quantum critical phenomena, *Phys. Rev. Lett.* **90**, 227902 (2003).

[17] A. Vilenkin and E. P. S. Shellard, *Cosmic Strings and Other Topological Defects*, Cambridge University Press (2000).