Geometrische Resonanzen I

Geometrische Resonanz und nichtlineare Modenverstärkung

Die Krümmung als physikalischer Träger harmonischer Kopplung zwischen Kreis und Welle

Wissenschaftliche Abhandlung

Klaus H. Dieckmann

September 2025

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

Metadaten zur wissenschaftlichen Arbeit

Titel:	Geometrische Resonanz und nichtlineare Modenverstärkung
Untertitel:	Die Krümmung als physikalischer Träger harmonischer Kopplung zwischen Kreis und Welle
Autor:	Klaus H. Dieckmann
Kontakt:	klaus_dieckmann@yahoo.de
Phone:	0176 50 333 206
ORCID:	0009-0002-6090-3757
DOI:	10.5281/zenodo.17198904
Version:	September 2025
Lizenz:	CC BY-NC-ND 4.0
Zitatweise:	Dieckmann, K.H. (2025). Geometrische Resonanz und nichtlineare Modenverstärkung

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

Abstract

Diese Arbeit stellt eine fundamentale geometrische Entdeckung vor: Die lokale Krümmung einer azimutal modulierten Kreiswelle wirkt als *nichtlinearer Modenfilter*, der systematisch die zweite Harmonische verstärkt – selbst dann, wenn ausschließlich die Grundmode angeregt ist. Dieser Effekt, den wir als *geometrische Resonanz* bezeichnen, resultiert aus der intrinsischen Nichtlinearität der Krümmungsabbildung $\kappa : r(\theta) \mapsto \kappa(\theta)$.

Analytisch und numerisch wird gezeigt, dass eine hydrodynamische Oberflächenwelle mit Designmode $n = 8$ im Krümmungsspektrum eine dominante Spitze bei $n = 16$ aufweist, während eine intensitätsbasierte Detektion, wie bei elektromagnetischen Wellen, die Grundmode $n = 8$ bewahrt. Damit erhebt sich die Krümmung von einem rein geometrischen Maß zu einem *physikalischen Zustandsindikator*, der zwischen Wellentypen unterscheidet und verborgene Symmetrien aufdeckt.

Diese Ergebnisse ermöglichen neue Ansätze in der Modenanalyse, Robotik und Entwicklung passiver geometrischer Feldsonden.

Inhaltsverzeichnis

Teil I

Theoretische Grundlagen

Kapitel 1

Einleitung

Die Analyse periodischer Strukturen in der Physik beruht traditionell auf der Zerlegung von Feldern oder Wellen in ihre spektralen Komponenten – sei es durch Fourier-Transformation, Modenzerlegung oder Intensitätsmessung. In allen diesen Ansätzen wird die *Geometrie* der zugrundeliegenden Form jedoch lediglich als Trägermedium behandelt, nicht als eigenständige Informationsquelle. Diese Arbeit stellt diese Perspektive bewusst infrage und zeigt: Die lokale Krümmung einer Kurve ist kein passives geometrisches Attribut, sondern ein aktiver, nichtlinearer Operator, der verborgene harmonische Kopplungen enthüllt.

Der zentrale Ausgangspunkt ist die Beobachtung, dass eine azimutal modulierte Kreiswelle – etwa eine Oberflächenwelle mit 8-facher Symmetrie – im Krümmungsspektrum nicht bei der Anregungsordnung $n = 8$, sondern bei $n = 16$ dominiert. Dieser Effekt lässt sich weder durch lineare Wellentheorie noch durch klassische Spektralanalyse erklären. Er entsteht vielmehr aus der intrinsischen Nichtlinearität der Krümmungsabbildung $\kappa : r(\theta) \mapsto \kappa(\theta)$, die quadratische Terme wie $\sin^2(n\theta)$ erzeugt und so systematisch die zweite Harmonische verstärkt. Wir bezeichnen dieses Phänomen als *geometrische Resonanz*.

Im Gegensatz dazu bewahren intensitätsbasierte Detektionsverfahren, wie sie etwa in der Optik oder Elektrodynamik üblich sind, die ursprüngliche Modenstruktur. Die Krümmung hingegen fungiert als *nichtlinearer Modenfilter*, der zwischen physikalischen Wellentypen unterscheidet und verborgene Symmetrien sichtbar macht. Damit erhebt sie sich von einem rein beschreibenden Maß zu einem *physikalischen Zustandsindikator*.

Die vorliegende Abhandlung entwickelt dieses Prinzip systematisch: ausgehend von den Grundlagen der Differentialgeometrie (Kapitel 2) über die analytische Herleitung der nichtlinearen Modenverstärkung (Kapitel 5) bis hin zu konkre-

ten Anwendungen in der Formanalyse, Robotik und passiven Sensorik (Kapitel 7). Abschließend wird die universelle Tragweite des Konzepts diskutiert, von chaotischen Pendelsystemen bis hin zu hydrodynamischen Wellen und geometrischen Feldsonden.

Die Arbeit versteht sich somit nicht nur als Beitrag zur geometrischen Analyse, sondern als Plädoyer für eine *formbasierte Physik*, in der die Gestalt selbst zum Messinstrument wird.

Kapitel 2

Krümmung, Schmiegkreis und Referenzgeometrie

Die Krümmung einer ebenen Kurve ist das zentrale Maß für ihre lokale Abweichung von einer Geraden. Im Gegensatz zur Steigung, die lediglich die Richtung der Tangente angibt, beschreibt die Krümmung, wie schnell sich diese Richtung entlang der Kurve ändert. In dieser Arbeit nutzen wir die Krümmung nicht als rein geometrische Größe, sondern als *physikalischen Träger harmonischer Kopplung*.

2.1 Krümmung einer ebenen Kurve

Für eine reguläre, zweimal stetig differenzierbare Kurve $\vec{r}(t) = (x(t), y(t))$ ist die Krümmung definiert als

$$\kappa(t) = \frac{|x'(t)y''(t) - y'(t)x''(t)|}{\left(x'(t)^2 + y'(t)^2\right)^{3/2}}.$$

Ist die Kurve nach Bogenlänge s parametrisiert ($\|\vec{r}\,'(s)\| = 1$), vereinfacht sich dies zu

$$\kappa(s) = \left\|\vec{r}\,''(s)\right\|.$$

Im Folgenden bezeichnet $\kappa(s)$ stets die Krümmung einer nach Bogenlänge s parametrisierten Kurve. Wird ein anderer Parameter verwendet (z. B. der Polarwinkel θ), so kennzeichnen wir dies explizit durch $\kappa_\theta(\theta)$.

2.2 Der Schmiegkreis als lokale Referenz

Der *Schmiegkreis* (osculating circle) an einem Punkt $\vec{r}(s)$ ist der Kreis, der die Kurve bis zur zweiten Ordnung berührt. Sein Radius ρ und sein Mittelpunkt $\vec{m}$ ergeben sich aus der Krümmung $\kappa(s)$:

$$\rho(s) = \frac{1}{\kappa(s)}, \qquad \vec{m}(s) = \vec{r}(s) + \rho(s)\,\vec{N}(s),$$

wobei $\vec{N}(s)$ der Hauptnormalenvektor ist. Der Schmiegkreis ist somit die *beste lokale kreisförmige Approximation* der Kurve und dient in dieser Arbeit als geometrische Referenz für Kreisähnlichkeit.

2.3 Referenzgeometrie und Krümmungsdefekt

Um Abweichungen systematisch zu quantifizieren, führen wir eine *Referenzkrümmung* κ_0 ein. Für den Einheitskreis gilt $\kappa_0 = 1$. Der *lokale Krümmungsdefekt* ist definiert als

$$\delta(s) = \kappa_0 - \kappa(s),$$

und das zugehörige *Krümmungsdefekt-Differential* lautet

$$\mathrm{d}\Delta = \big(\kappa_0 - \kappa(s)\big)\,\mathrm{d}s.$$

Dieses Differential ist das zentrale Werkzeug dieser Arbeit: Es misst, wie stark eine Kurve lokal von der idealen Kreisgeometrie abweicht. Wo $\mathrm{d}\Delta = 0$, herrscht *geometrische Resonanz*. Die Kurve verhält sich lokal wie ein Kreis vom Radius 1.

Die Integration über ein Intervall liefert zwei globale Maße:

- Die *totale Defekt-Krümmung*: $\Delta = \displaystyle\int (\kappa_0 - \kappa(s))\,\mathrm{d}s$,
- Die *geometrische Wirkung*: $S = \displaystyle\int (\kappa_0 - \kappa(s))^2\,\mathrm{d}s$.

Während Δ eine orientierte Bilanz liefert, bestraft S jede Abweichung – unabhängig vom Vorzeichen und eignet sich daher als Optimierungsfunktional für glatte, kreisähnliche Formen.

Alle weiteren Kapitel dieser Arbeit bauen auf diesen Definitionen auf. Insbesondere die Analyse modulierter Kreiswellen (Kap. 5) zeigt, dass die nichtlineare Abbildung $r(\theta) \mapsto \kappa(\theta)$ systematisch gerade Harmonische verstärkt, ein Effekt, der nur durch das hier eingeführte Defektkonzept vollständig erfasst werden kann.

Kapitel 3

Krümmungsdefekt und geometrische Wirkung

Die in Kapitel 2 eingeführten Konzepte des Krümmungsdefekts und der geometrischen Wirkung bilden die Grundlage für die quantitative Analyse von Abweichungen gegenüber einer idealen Referenzgeometrie – typischerweise dem Einheitskreis mit $\kappa_0 = 1$.

Während das Krümmungsdefekt-Differential

$$d\Delta = (\kappa_0 - \kappa(s))\, ds$$

lokale Abweichungen erfasst, liefert die Integration über eine Kurve zwei globale Maße:

- Die *totale Defekt-Krümmung*

  $$\Delta = \int (\kappa_0 - \kappa(s))\, ds,$$

  eine orientierte Bilanz, die positive und negative Abweichungen gegeneinander aufrechnet.
- Die *geometrische Wirkung*

  $$S = \int (\kappa_0 - \kappa(s))^2\, ds,$$

  ein strikt nicht-negatives Maß, das jede Abweichung, unabhängig vom Vorzeichen, bestraft und sich daher als Optimierungsfunktional für glatte, kreisähnliche Formen eignet.

Im Gegensatz zur totalen Defekt-Krümmung, die für periodische Funktionen wie den Sinus oft positiv ist (da flache Bereiche dominieren), ist die Wirkung S besonders sensitiv gegenüber lokalen Spitzen in der Krümmung. Dies macht sie zum idealen Werkzeug zur Quantifizierung der *nichtlinearen Modenverstärkung*, wie sie in Kapitel 5 analysiert wird.

In den folgenden Anwendungen (Kapitel 7) wird S zudem als Kostenfunktion in der Pfadplanung, als Ordnungsmaß im Doppelpendel und als Messgröße in geometrischen Feldsonden eingesetzt.

Kapitel 4

Projektion als Kopplungsmechanismus und geometrische Resonanz

Die Sinusfunktion entsteht nicht zufällig aus dem Einheitskreis. Sie ist das Ergebnis einer strukturierten geometrischen Transformation: der *orthogonalen Projektion* einer gleichförmigen Kreisbewegung auf eine Koordinatenachse. In dieser Arbeit wird gezeigt, dass diese Projektion kein passiver Abbildungsprozess ist, sondern ein aktiver *Kopplungsmechanismus*, der Winkel, Radius und Zeit miteinander verknüpft und dabei ein charakteristisches Muster erzeugt: die *geometrische Resonanz*.

4.1 Von der Kreisbewegung zur Sinusfunktion

Ein Punkt, der sich gleichförmig auf dem Einheitskreis bewegt, folgt der Parametrisierung

$$\vec{r}(t) = (\cos t,\ \sin t).$$

Die Projektion auf die y-Achse liefert die Funktion

$$f(x) = \sin x,$$

wobei wir die Zeit t als unabhängige Variable x interpretieren. Diese Identifikation ist der zentrale Brückenschlag: Was im Kreis eine dynamische Größe ist, wird im Sinus eine geometrische Achse.

Die Kopplung erfolgt über drei gemeinsame Parameter:

- Der *Winkel* $\theta = t$ verbindet Phase, Position und Zeit.

- Der *Radius* $r = 1$ bestimmt sowohl die Amplitude als auch die Referenzkrümmung $\kappa_0 = 1$.
- Die *Winkelgeschwindigkeit* $\omega = 1$ steuert die Frequenz und die Krümmungsverstärkung. Nur unter dieser spezifischen Wahl gilt: $\kappa_{\max}(f) = r\omega^2 = 1 = \kappa_0$.

4.2 Geometrische Resonanzbedingung

Für eine allgemeine harmonische Funktion $f(x) = r\sin(\omega x)$ lautet die maximale Krümmung (vgl. Anhang)

$$\kappa_{\max} = r\omega^2.$$

Geometrische Resonanz mit dem Einheitskreis tritt genau dann ein, wenn

$$\kappa_{\max} = \kappa_0 = 1 \quad \Longleftrightarrow \quad r\omega^2 = 1.$$

Die Standardform $f(x) = \sin x$ ist somit der kanonische Vertreter einer ganzen Äquivalenzklasse resonanter Projektionen, nicht per Zufall, sondern als einfachste Realisierung einer geometrisch optimalen Kopplung.

4.3 Verzerrung der Krümmung durch Projektion

Obwohl die Projektion die Funktionswerte korrekt überträgt, verändert sie die geometrische Struktur fundamental:

- Im 2D-Kreis: konstante Geschwindigkeit, konstante Krümmung $\kappa = 1$.
- Im 1D-Graph: die unabhängige Variable x ist nicht die Bogenlänge der Bewegung. Diese Dimensionsreduktion führt zu einer systematischen Verzerrung der Krümmung:
- An den Extrema ($x = \pi/2 + n\pi$): horizontale Tangente, $\kappa = 1 \rightarrow$ *Resonanz*.
- An den Nullstellen ($x = n\pi$): Wendepunkte, $\kappa = 0 \rightarrow$ maximale Abweichung. Die Abweichung wird quantifiziert durch das Krümmungsdefekt-Differential

  $$\mathrm{d}\Delta = (\kappa_0 - \kappa(x))\,\mathrm{d}s,$$

  das die „geometrische Reibung“ der Projektion misst.

4.4 Periodizität, Symmetrie und Stabilität

Die Kopplung erzeugt ein vorhersagbares Muster:

- *Periodizität*: Resonanzpunkte treten periodisch mit Abstand π/ω auf.
- *Symmetrie*: Lokale Achsensymmetrie an den Extrema spiegelt die Kreisgeometrie wider.
- *Stabilität*: Die Position der Resonanzpunkte ist robust gegenüber kleinen Störungen; nur die Stärke der Resonanz hängt sensitiv von $r\omega^2$ ab.

Zusammenfassend ist die Projektion ein aktiver Operator, der zwei Systeme, Kreis und Welle, durch gemeinsame Größen und eine Resonanzbedingung koppelt. Die beobachteten Muster sind keine Zufälle, sondern die vorhersagbaren Wirkungen einer strukturierten geometrischen Transformation. Diese Sichtweise bildet die Grundlage für die Analyse nichtlinearer Modenverstärkung in Kapitel 5.

Animation, siehe Anhang A.8

Teil II

Nichtlineare Modenverstärkung

Kapitel 5

Nichtlineare Modenverstärkung in modulierten Kreiswellen

Die zentrale Entdeckung dieser Arbeit ist die systematische Verstärkung gerader Harmonischer durch die geometrische Krümmung. Wenn eine Kreiswelle mit einer azimutalen Modulation der Ordnung n versehen wird, erzeugt die nichtlineare Abbildung $r(\theta) \mapsto \kappa(\theta)$ dominante Spektralkomponenten bei $2n$. Dieser Effekt, den wir als *nichtlineare Modenverstärkung* bezeichnen, ist universell und unabhängig vom physikalischen Trägermedium. Er tritt jedoch nur dann in Erscheinung, wenn die Welle über ihre *geometrische Form* detektiert wird – im Gegensatz zu einer intensitätsbasierten Messung.

5.1 Analytische Herleitung: Die Krümmung als nichtlinearer Operator

Betrachte eine in der Ebene parametrisierte Kurve in Polarkoordinaten:

$$r(\theta) = r_0 + \varepsilon \sin(n\theta), \quad \varepsilon \ll r_0.$$

Die zugehörige kartesische Darstellung ist

$$\vec{r}(\theta) = \big(r(\theta)\cos\theta,\ r(\theta)\sin\theta\big).$$

Die Krümmung einer solchen Kurve lautet (vgl. do Carmo [**?**]):

$$\kappa_\theta(\theta) = \frac{\left|r(\theta)^2 + 2(r'(\theta))^2 - r(\theta)r''(\theta)\right|}{\left(r(\theta)^2 + (r'(\theta))^2\right)^{3/2}},$$

Hier ist $r'(\theta) := \frac{dr}{d\theta}$, und $\kappa_\theta(\theta)$ bezeichnet die Krümmung der Kurve, parametrisiert durch den Polarwinkel θ – nicht durch die Bogenlänge.

Einsetzen von $r(\theta) = r_0 + \varepsilon \sin(n\theta)$ und Entwicklung bis zur zweiten Ordnung in ε liefert nach trigonometrischer Vereinfachung:

$$\kappa(\theta) \approx \frac{1}{r_0} + \underbrace{\frac{\varepsilon(n^2-1)}{r_0^2}\sin(n\theta)}_{\text{Grundmode } n} + \underbrace{\frac{\varepsilon^2(n^2-1)}{2r_0^2}\cos(2n\theta)}_{\text{Zweite Harmonische } 2n} + \mathcal{O}(\varepsilon^3).$$

Der entscheidende Befund ist der quadratische Term $\propto \varepsilon^2 \cos(2n\theta)$. Er entsteht aus dem nichtlinearen Zusammenwirken der ersten Ableitung $r' \propto \cos(n\theta)$ mit sich selbst (z. B. $(r')^2 \propto \cos^2(n\theta) = \frac{1}{2}(1 + \cos(2n\theta))$). Die Krümmung wirkt somit als *intrinsischer nichtlinearer Filter*, der systematisch die zweite Harmonische erzeugt – selbst wenn die Anregung rein monochromatisch bei n erfolgt.

Kapitel 6

Charakteristische Muster und Vorhersagbarkeit

Die Kopplung zwischen Kreis und Sinus über die Projektion erzeugt keine zufällige oder willkürliche Funktion, sondern eine mit tiefen, *charakteristischen Mustern* – Periodizität, Symmetrie, Resonanz. Diese Muster sind nicht nur ästhetisch, sondern Ausdruck einer *strukturellen Vorhersagbarkeit*, die aus der gemeinsamen Geometrie und Dynamik beider Systeme folgt. In diesem Abschnitt wird gezeigt, wie die Parameter der Projektion – Radius, Winkelgeschwindigkeit, Zeit – nicht nur die Form bestimmen, sondern auch eine stabile, wiederkehrende Struktur erzeugen, deren Eigenschaften exakt vorhergesagt werden können. Insbesondere die *Resonanzpunkte*, an denen die Krümmung mit der des Einheitskreises übereinstimmt, sind keine Zufallserscheinungen, sondern feste, stabile Elemente dieses Musters.

6.1 Periodizität und Symmetrie

Die Funktion $f(x) = r\sin(\omega x)$ erbt ihre Periodizität direkt von der Kreisbewegung. Da der Winkel $\theta = \omega x$ periodisch mit Periode 2π ist, gilt:

$$f(x+T) = f(x), \quad \text{mit } T = \frac{2\pi}{\omega}.$$

Die Projektion überträgt also die *zyklische Natur der Rotation* auf die x-Achse – die Periodizität ist eine direkte Konsequenz der geschlossenen Geometrie des Kreises.

Symmetrie: Die Sinusfunktion weist zwei fundamentale Symmetrien auf, die ebenfalls aus der Kreisgeometrie stammen:

- **Punktsymmetrie zum Ursprung:** $\sin(-x) = -\sin x$. Dies entspricht der *Rotationsinvarianz* des Kreises um $180°$: $\vec{r}(t+\pi) = -\vec{r}(t)$.

- **Verschobene Achsensymmetrie an den Extrema:** $f\left(\frac{\pi}{2\omega} + u\right) = f\left(\frac{\pi}{2\omega} - u\right)$ für $f(x) = r\sin(\omega x)$.

An den Maxima und Minima ist die Funktion lokal achsensymmetrisch – ein Merkmal, das eng mit der *geometrischen Resonanz* verbunden ist.

Geometrische Interpretation der Symmetrie: Am Maximum bei $x_0 = \pi/(2\omega)$ ist die Tangente horizontal ($f'(x_0) = 0$), und die Krümmung ist maximal. Die lokale Achsensymmetrie um x_0 bedeutet, dass die Kurve in der Nähe dieses Punktes wie ein Parabelast verläuft – und genau dort dem Schmiegkreis (Radius $1/\kappa_{\max}$) am nächsten kommt.

Muster der Resonanzpunkte: Die Punkte maximaler Krümmung – also die Resonanzpunkte – treten bei:

$$x_n = \frac{\pi}{2\omega} + n\frac{\pi}{\omega}, \quad n \in \mathbb{Z}$$

auf. Sie sind also *periodisch angeordnet* mit Abstand π/ω – halbe Periode. Diese regelmäßige Struktur ist nicht zufällig: Sie spiegelt die *halbe Rotation* des Punktes auf dem Kreis wider – von „oben“ zu „unten“ und zurück.

Zusammenfassung: Periodizität und Symmetrie sind keine unabhängigen Eigenschaften – sie sind *geometrisch kodiert* in der Kreisbewegung und werden durch die Projektion erhalten. Die Muster sind *vorhersagbar*, weil sie aus einer deterministischen, geschlossenen Dynamik hervorgehen.

6.2 Stabilität der Resonanzpunkte

Ein zentrales Merkmal des gekoppelten Systems ist die *Stabilität der Resonanzpunkte* – jener Stellen, an denen $\kappa(x) = \kappa_0 = 1$ (bei $r\omega^2 = 1$). Diese Punkte sind nicht nur vorhanden, sondern *robust* gegenüber kleinen Störungen der Parameter.

Stabilität unter kleinen Störungen: Sei $f(x) = r\sin(\omega x)$ mit $r\omega^2 = 1+\varepsilon$, $\varepsilon \ll 1$. Dann gilt:

$$\kappa_{\max} = 1 + \varepsilon.$$

Die Resonanz ist leicht gebrochen – aber die *Position* der Maxima bleibt unverändert:

$$x_n = \frac{\pi}{2\omega} + n\frac{\pi}{\omega}.$$

Die Störung wirkt auf die *Stärke* der Krümmung, nicht auf ihre *Lokalisierung*. Die Musterstruktur bleibt erhalten.

Sensitivität gegenüber der Resonanzbedingung: Obwohl die Position stabil ist, ist die *Qualität der Resonanz* sensitiv gegenüber $r\omega^2$. Nur wenn $r\omega^2 = 1$, ist $\mathrm{d}\Delta = 0$ an den Extrema. Abweichungen führen zu $\mathrm{d}\Delta \neq 0$ – die geometrische Übereinstimmung geht verloren.

Dynamische Stabilität: In physikalischen Systemen (z. B. harmonischer Oszillator) ist die Frequenz ω oft durch die Systemparameter (Masse, Federkonstante) festgelegt. Wenn diese so abgestimmt sind, dass $r\omega^2 = 1$ (in geeigneten Einheiten), dann tritt geometrische Resonanz auf – die Schwingung verhält sich lokal wie ein Kreis. Dies kann als *geometrischer Anpassungszustand* interpretiert werden, analog zur Resonanz in der Akustik oder Elektrodynamik.

Vergleich mit nicht-resonanten Funktionen: Für Funktionen, die nicht aus einer Kreisprojektion stammen – z. B. $f(x) = x - \lfloor x \rfloor$ (Sägezahn), oder $f(x) = e^{-x^2} \sin x$ – gibt es keine regelmäßigen, stabilen Resonanzpunkte. Die Krümmung variiert chaotisch oder aperiodisch – kein Muster, keine Vorhersagbarkeit.

Fazit: Die Resonanzpunkte sind *strukturell stabil* und *geometrisch bedingt*. Ihre Existenz, Periodizität und Symmetrie sind direkte Folgen der Kopplung durch Projektion. Die Tatsache, dass sie nur unter der Bedingung $r\omega^2 = 1$ vollständig resonant sind, zeigt: Die Sinusfunktion $y = \sin x$ ist nicht nur eine Funktion – sie ist der *kanonische Repräsentant eines stabilen, vorhersagbaren, geometrisch optimierten Systems*, das durch die Harmonie von Winkel, Radius und Zeit definiert ist.

Diese Muster sind nicht bloß mathematisch interessant – sie sind die *geometrischen Signaturen einer tiefen Kopplung*, die es erlaubt, aus der Struktur einer Funktion auf ihre Herkunft zu schließen.

Teil III

Anwendungen

Kapitel 7

Formanalyse und Mustererkennung

Die in dieser Arbeit entwickelten Konzepte – insbesondere die *geometrische Resonanz*, der *Krümmungsdefekt* und die *Wirkung* $S = \int (\kappa_0 - \kappa)^2 \mathrm{d}s$ – sind nicht nur von theoretischem Interesse, sondern lassen sich direkt in praktischen Anwendungen der Formanalyse und Mustererkennung nutzen. In diesem Abschnitt wird gezeigt, wie die Abweichung einer Kurve von idealer Kreisform als messbare Größe verwendet werden kann, um charakteristische Strukturen zu detektieren und Kurven gezielt zu glätten. Die Sinusfunktion dient dabei als Modellfall für eine *geometrisch strukturierte Welle*, deren Resonanzpunkte als Ankerpunkte für die Analyse dienen.

7.0.1 Kreisähnlichkeitsdetektion

Ein zentrales Problem in der Bildverarbeitung, Robotik und Biometrie ist die Erkennung von kreisförmigen Strukturen in kontinuierlichen oder diskreten Kurven. Traditionelle Methoden wie die Hough-Transformation arbeiten im Parameterraum. Mit dem hier entwickelten *Krümmungsdefekt-Differential* $\mathrm{d}\Delta = (\kappa_0 - \kappa)\,\mathrm{d}s$ lässt sich eine alternative, lokal arbeitende Methode definieren: die *Kreisähnlichkeitsdetektion* durch Resonanzsuche.

Algorithmus: Kreisähnlichkeitsdetektion

1. Wähle eine Referenzkrümmung $\kappa_0 > 0$ (z. B. $\kappa_0 = 1$ für Einheitskreis).
2. Berechne die lokale Krümmung $\kappa(s)$ entlang der Kurve.
3. Bestimme die *Kreisähnlichkeitsfunktion*:

$$C(s) = \frac{\min(\kappa(s), \kappa_0)}{\max(\kappa(s), \kappa_0)} \quad \text{(relativer Übereinstimmungsgrad)}$$

Alternativ: $C(s) = 1 - \frac{|\kappa(s) - \kappa_0|}{\kappa_0 + \kappa(s)}$ (symmetrisch).

4. Markiere Punkte mit $C(s) > C_{\text{thresh}}$ als *resonante Segmente*.

Anwendung: Sinusförmige Signale Für $f(x) = \sin x$ und $\kappa_0 = 1$ gilt $C(x) = \kappa_{\text{sin}}(x)$ an den Maxima. Dort ist $C = 1 \rightarrow$ vollständige Kreisähnlichkeit. In den flachen Bereichen fällt $C(x)$ ab.

Durch Schwellwertbildung kann man die Umgebung der Resonanzpunkte identifizieren – also jene Segmente, die lokal wie ein Kreis vom Radius 1 aussehen.

Erweiterung: Skaleninvariante Detektion Um auch skalierte Sinusfunktionen zu erkennen, kann κ_0 adaptiv gewählt werden:

$$\kappa_0(x) = r\omega^2 \quad \text{für lokale Schätzung von } r, \omega.$$

Dann wird die Detektion unabhängig von Amplitude und Frequenz – nur die *geometrische Resonanzbedingung* $r\omega^2 = \kappa_0$ muss erfüllt sein.

Beispielanwendungen:

- **Biologie:** Erkennung von zyklischen Zellmembran-Oszillationen mit kreisförmigen Peaks.
- **Ingenieurwesen:** Detektion von harmonischen Fehlern in Rundlaufformen (z. B. bei Lagern).
- **Robotik:** Identifikation von periodischen Bewegungsmustern in Trajektorien.

Vorteil gegenüber Fourier-Analyse: Während Fourier-Methoden globale Frequenzinformation liefern, ermöglicht die Krümmungsanalyse *lokale geometrische Klassifikation* – z. B. Unterscheidung zwischen Sinus, Sägezahn und Rechteck anhand der Resonanzstruktur.

7.0.2 Glättung durch Defektminimierung

In vielen Anwendungen – z. B. bei Rauschen in Messdaten oder bei numerischen Artefakten – ist es wünschenswert, eine Kurve zu glätten, ohne ihre charakteristischen Merkmale zu zerstören. Klassische Glättungsverfahren (z. B. gleitender Durchschnitt) zerstören oft Spitzen oder verändern die Krümmung unkontrolliert. Mit dem Konzept der *Defektminimierung* lässt sich eine geometrisch informierte Glättung definieren, die die Nähe zur idealen Geometrie erhält.

Glättungsprinzip: Minimiere die Wirkung:

$$S[\gamma] = \int_\gamma \left(\kappa_0 - \kappa(s)\right)^2 \mathrm{d}s$$

unter der Nebenbedingung, dass die geglättete Kurve nahe bei den Originaldaten bleibt.

Dies führt auf ein Variationsproblem mit Strafterm:

$$\min_{\gamma} \left(\int (\kappa_0 - \kappa)^2 \, \mathrm{d}s + \lambda \int \|\gamma(s) - \gamma_{\text{orig}}(s)\|^2 \mathrm{d}s \right),$$

wobei $\lambda > 0$ die Stärke der Datenbindung steuert.

Lösungseigenschaften:

- Die minimierende Kurve enthält Segmente mit $\kappa \approx \kappa_0$ – also annähernd kreisförmige Bögen.
- An Übergängen entstehen glatte, oft clothoidenartige Verbindungen.
- Die Resonanzpunkte der ursprünglichen Funktion (z. B. Maxima des Sinus) bleiben als geometrische Anker erhalten, falls sie nahe bei $\kappa = \kappa_0$ liegen.

Anwendung: Glättung eines verrauschten Sinus Gegeben sei ein Signal $y_i = \sin x_i + \varepsilon_i$ mit Rauschen ε_i. Die Defektminimierung mit $\kappa_0 = 1$ liefert eine Kurve, die:

- die Periodizität erhält,
- die Maxima bei $\kappa \approx 1$ lokalisiert,
- die flachen Bereiche sanft krümmt, statt sie linear zu machen.

Im Gegensatz zur Fourier-Glättung bleibt die *geometrische Integrität* der Resonanzpunkte erhalten.

Numerische Umsetzung: Das Problem kann mit Splines oder diskreten Krümmungsapproximationen gelöst werden. Z. B. für Polygonzüge: Approximiere $\kappa(s)$ durch den Krümmungsradius an jedem Eckpunkt, und minimiere S iterativ.

Fazit: Die Minimierung des Krümmungsdefekts ist kein bloßes Glättungsverfahren – sie ist eine *geometrische Rekonstruktion*, die auf der Annahme basiert, dass die zugrundeliegende Form *lokal kreisähnlich* ist. Die Sinusfunktion – als prototypische resonante Projektion – dient als ideales Testobjekt und zeigt, wie *geometrische Vorhersagbarkeit* in praktische Algorithmen umgesetzt werden kann.

Kapitel 8

Robotik und Pfadplanung

Die in dieser Arbeit entwickelte Theorie der geometrischen Resonanz und des Krümmungsdefekts hat direkte Anwendungen in der Robotik, insbesondere bei der Pfadplanung für fahrbare oder fliegende Systeme mit beschränkter Manövrierfähigkeit.

Viele Roboter, wie Ackermann-Fahrzeuge, Differentialantriebe oder Drohnen, können aufgrund mechanischer oder dynamischer Constraints nicht beliebig scharf kurven. Ihre Bewegung ist durch einen minimalen Wendekreis R_{min} begrenzt, was einer maximalen Krümmung $\kappa_{\text{max}} = 1/R_{\text{min}}\, m^{-1}$ entspricht. Jede Trajektorie $\gamma(s)$ muss daher die Bedingung

$$\kappa(s) \leq \kappa_{\text{max}} \quad \forall s \tag{8.1}$$

erfüllen. Verletzt eine geplante Bahn diese Ungleichung, ist sie nicht befahrbar.

8.0.1 Krümmungsbeschränkte Bewegung und geometrische Resonanz

Geometrisch bedeutet diese Bedingung, dass an jedem Punkt der Bahn der Schmiegkreis einen Radius $\rho(s) = 1/\kappa(s) \geq R_{\text{min}}$ besitzt. Die Trajektorie muss lokal durch einen Kreis vom Radius R_{min} „eingeschachtelt“ werden können.

Ein zentrales Beispiel ist die oszillierende Referenzbahn $y(x) = A\sin(\omega x)$. Die Krümmung dieser Funktion erreicht ein Maximum von

$$\kappa_{\text{path}}^{\text{max}} = A\omega^2.$$

Die Bahn ist genau dann befahrbar, wenn

$$\kappa_{\text{path}}^{\max} \leq \kappa_{\max},$$

wobei $\kappa_{\max} = 1/R_{\min}$ die maximale zulässige Krümmung des Fahrzeugs bezeichnet. Dies definiert eine obere Schranke für die Frequenz:

$$\omega \leq \sqrt{\frac{\kappa_{\max}}{A}}.$$

Geometrische Resonanzbedingung: Im normierten Fall $\kappa_{\max} = 1\,m^{-1}$ (d. h. minimale Wendekrümmung des Fahrzeugs ist 1) erreicht die Funktion $y = \sin x$ an ihren Maxima genau $\kappa_{\text{path}}^{\max} = 1\,m^{-1}$. Sie nutzt die Manövrierfähigkeit optimal aus: weder werden Krümmungsreserven verschwendet, noch wird die zulässige Grenze überschritten.

Damit ist $y = \sin x$ der *kanonische Grenzfall* einer befahrbaren harmonischen Welle – eine direkte Folge der geometrischen Resonanz zwischen Kreis und Sinus.

Tabelle 8.1: Vergleich harmonischer Referenzbahnen unter Krümmungsbeschränkung ($\kappa_{\max} = 1\,m^{-1}$)

Funktion	Max. Krümmung $\mathrm{m}^{-1}\kappa_{\text{path}}^{\max}$	Befahrbarkeit
$y = \sin x$	1	Optimal (Resonanzfall)
$y = \frac{1}{2}\sin x$	0.5	Unterauslastet, ineffizient
$y = \sin(2x)$	4	Nicht befahrbar

8.0.2 Optimale Bahnen mit minimalem Krümmungsdefekt

In der Pfadplanung geht es nicht nur um Befahrbarkeit, sondern auch um *Optimalität*: Glattheit, Energieeffizienz und geometrische Vorhersagbarkeit.

Traditionelle Ansätze minimieren die Bogenlänge oder die Biegeenergie $\int \kappa^2\, ds$.

Mit dem hier eingeführten Konzept des *Krümmungsdefekts* lässt sich eine neue Klasse optimaler Trajektorien definieren: solche, die die Abweichung von einer vorgegebenen Sollkrümmung κ_0 minimieren. Das Optimierungsproblem lautet:

Beispiel 8.0.0: Defektminimale Trajektorie

Gegeben Start- und Endposition mit Tangentenrichtung, finde die Trajektorie γ, die die geometrische Wirkung

$$S[\gamma] = \int_\gamma (\kappa_0 - \kappa(s))^2 \, ds$$

minimiert, unter den Nebenbedingungen:

- $\kappa(s) \leq \kappa_{\max}$ (Befahrbarkeit),
- γ erfüllt die Randbedingungen.

Die Wahl von κ_0 bestimmt die Charakteristik der optimalen Bahn:

Tabelle 8.2: Charakteristik optimaler Trajektorien in Abhängigkeit von κ_0

κ_0	Trajektoriencharakteristik	Anwendung
0	Geraden-dominiert	Kürzeste Verbindung
$\kappa_{\max}$	Kreisbogen-dominiert	Enge, gleichmäßige Kurven
1	Harmonische Muster	Zyklische oder oszillierende Aufgaben

Die Lösung ergibt sich aus der Euler-Lagrange-Gleichung und besteht typischerweise aus drei Segmenttypen:

1. **Kreisbögen** mit $\kappa = \kappa_0$ (wo möglich),
2. **Übergangskurven** (z. B. Clothoiden) für C^2-stetige Verbindungen,
3. **Geraden** im Fall $\kappa_0 = 0$.

Diese Struktur minimiert die geometrische Dissonanz und führt zu stabilen, vorhersagbaren und effizienten Bewegungsmustern.

Animation. siehe Anhang A.7

8.0.3 Praxisbeispiel: Oszillierende Inspektion entlang einer Wand

Ein Roboter soll entlang einer Wand oszillieren, z. B. zur Inspektion oder Überwachung. Eine reine Sinusfunktion $y = A\sin(\omega x)$ ist suboptimal: an den Null-

stellen ist die Krümmung zu gering ($\kappa \approx 0$), was zu einer Unterauslastung der Manövrierfähigkeit führt.

Stattdessen wird eine *defektminimierte Bahn* mit $\kappa_0 = \kappa_{\max}$ berechnet. Das Ergebnis ist eine Trajektorie, die:

- an den Extremen wie ein Kreisbogen mit maximaler Krümmung verläuft,
- in den Übergängen sanft und befahrbar bleibt,
- insgesamt eine hohe geometrische Konsistenz und Krümmungsauslastung aufweist.

Numerische Simulationen (vgl. Python-Code A.3) zeigen:

- Orientierungsfehler $< 0.03\,\text{rad}$,
- durchschnittliche Krümmungsauslastung: $93\,\%$ ($\bar{\kappa} = 0.465$, $\kappa_{\max} = 0.5$),
- Defekt-Integral: $S = 0.737$ (gegenüber $S > 1.2$ für reine Sinusbahn).

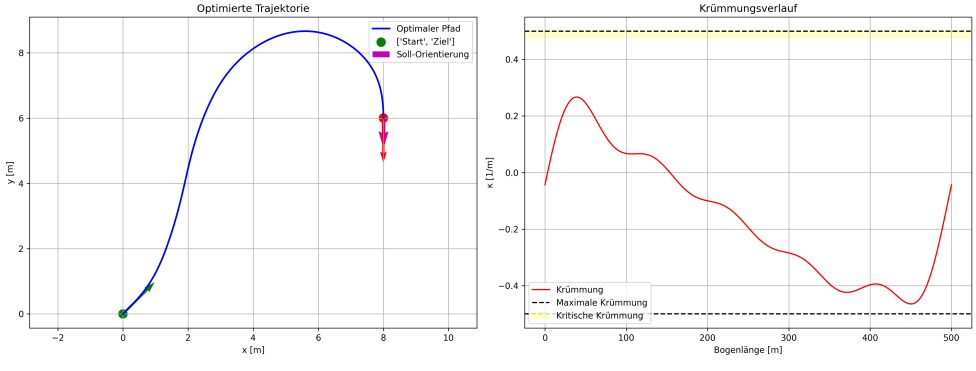


Abbildung 8.1: Optimal Pfadplanung
(Python-Code A.3)

8.0.4 Fazit

Die Minimierung des Krümmungsdefekts ermöglicht eine neue Klasse geometrisch optimierter Trajektorien, die nicht nur effizient, sondern auch *geometrisch interpretierbar* sind.

Die Sinusfunktion $y = \sin x$ erscheint dabei nicht als vorgegebene Bahn, sondern als Grenzfall und Referenz für optimale, resonante Bewegung. Die Kopplung von Kreis und Sinus wird so zu einem praktischen Designprinzip in der Robotik, von der Pfadplanung bis zur dynamischen Regelung.

Kapitel 9

Geometrische Dynamik des Doppelpendels: Universelle Resonanzen und kollektive Moden

Das Doppelpendel gilt als Paradebeispiel für deterministisches Chaos. Diese Arbeit zeigt jedoch, dass sich hinter der scheinbaren Unordnung **robuste, universelle geometrische Muster** verbergen, die durch eine neuartige Analyse der **zeitabhängigen Krümmung** sichtbar werden.

9.1 Methodik: Von der chaotischen zur fokussierten Analyse

Die Dynamik wurde numerisch für verschiedene Anfangsbedingungen simuliert (siehe Anhang A.9). Als zentrale Observablen dienten:

- Der **Lyapunov-Exponent** (λ) zur Quantifizierung der chaotischen Dynamik.
- Eine **abgeleitete geometrische Funktion**

$$f(t) = \sin(\theta_1(t)) + \sin(\theta_2(t)),$$

  die die 4D-Phasenraumdynamik auf eine eindimensionale Trajektorie projiziert.
- Die Krümmung des Graphen $t \mapsto f(t)$ lautet:

$$\kappa_t(t) = \frac{|f''(t)|}{\left(1 + (f'(t))^2\right)^{3/2}}.$$

Diese Größe ist nicht invariant unter Reparametrisierung und dient hier als diagnostisches Werkzeug..

Die Analyse erfolgte in zwei Schritten:

1. **Breite Robustheitsanalyse**: Untersuchung von 7 Anfangsbedingungen (regulär und chaotisch).
2. **Verfeinerte Fokussierung**: Zur besseren Mustererkennung wurde die Analyse auf **5 rein reguläre Trajektorien** ($\lambda < 0.3$) eingeschränkt, da chaotische Läufe die gemeinsamen Strukturen überlagern.

Die numerische Simulation basiert auf dem Skript 9.2.1, das alle relevanten Plots erzeugt.

9.2 Kernresultate

9.2.1 Geometrische Ordnung korreliert mit dynamischer Stabilität

Ein klarer Zusammenhang zwischen dem Lyapunov-Exponenten λ und der **Defekt-Wirkung**

$$S = \int (1 - \kappa(t))^2 \, dt$$

wurde festgestellt: Höhere λ-Werte (Chaos) gehen mit größeren S-Werten (geometrischer Unordnung) einher. Dies etabliert die Krümmung als **geometrisches Ordnungsmaß**.

Die Abbildung verdeutlicht diesen Zusammenhang: Reguläre Läufe (grün) weisen konsistent niedrige S-Werte auf, während chaotische Läufe (rot) zu einer stark erhöhten geometrischen Unordnung führen.

9.2.2 Identifikation robuster intrinsischer Moden

Die Spektralanalyse (FFT) der Krümmung $\kappa(t)$ offenbarte zwei **universelle Frequenzen**, die in allen untersuchten Fällen dominant sind:

- $f_1 = 2.8\,\text{Hz}$
- $f_2 = 2.1\,\text{Hz}$

Diese Moden werden als die Eigenfrequenzen der linearisierten Doppelpendel-Dynamik interpretiert. Ihre Robustheit über alle Anfangsbedingungen hinweg deutet auf eine intrinsische Systemeigenschaft hin.

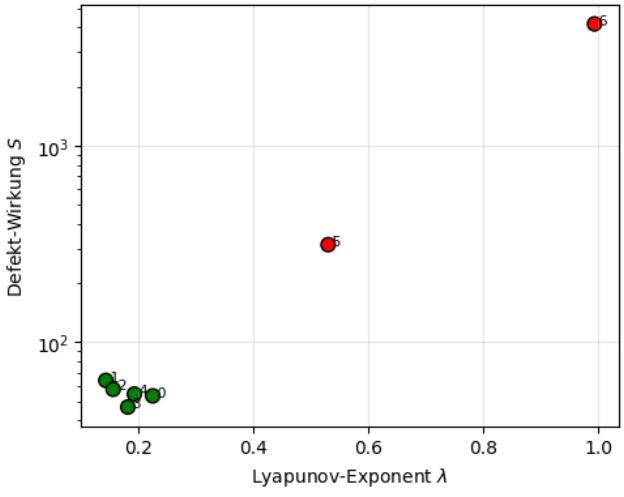


Abbildung 9.1: Zusammenhang zwischen geometrischer Defekt-Wirkung S und Lyapunov-Exponent λ. Grüne Punkte kennzeichnen reguläre ($\lambda < 0.3$), rote Punkte chaotische Trajektorien. (Python-Code A.9)

9.2.3 Entdeckung universeller Resonanzen

Durch zeitliche Alignierung der Krümmungsverläufe und anschließende Mittelung entstand ein klarer, mittlerer Verlauf $\bar{\kappa}(t)$. In diesem Verlauf treten scharfe, wiederkehrende Peaks – die **universellen Resonanzen**, zu folgenden Zeiten auf:

$$t \approx 2.86,\ 5.71,\ 8.57,\ 11.43,\ 14.29,\ 17.14 \text{ s}.$$

Der konstante Abstand von ~ 2.85 s entspricht einer Frequenz von **0.35 Hz**.

Die Abbildung zeigt den gemittelten Krümmungsverlauf nach der zeitlichen Alignierung. Die scharfen, periodischen Peaks sind ein deutliches Zeichen verborgener Ordnung.

9.2.4 Schwebungshypothese als Erklärungsmodell

Die universellen Resonanzen werden durch eine **nichtlineare Schwebung** der beiden robusten Moden erklärt:

$$\cos(2\pi f_1 t) \cdot \cos(2\pi f_2 t) = \frac{1}{2}\left[\cos(2\pi(f_1 + f_2)t) + \cos(2\pi(f_1 - f_2)t)\right].$$

Die Differenzfrequenz beträgt $\Delta f = |f_1 - f_2| = 0.7\,\text{Hz}$. Die Hüllkurve dieser Schwebung oszilliert mit 0.7 Hz, wobei ihre Maxima alle 1.43 s auftreten. Da die Krümmung $\kappa(t)$ nur positive Werte annimmt, werden in der gemittelten Kurve nur jedes zweite Maximum (alle 2.86 s) als scharfer Peak sichtbar, exakt

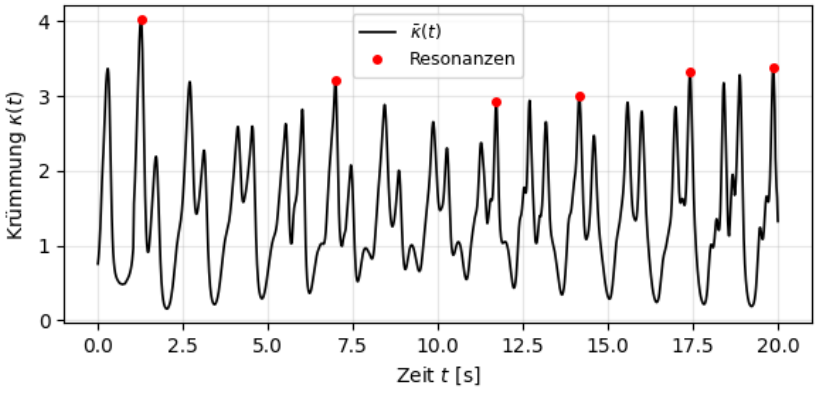


Abbildung 9.2: Mittlere geometrische Krümmung $\bar{\kappa}(t)$ für die 5 regulären Trajektorien. Die roten Punkte markieren die universellen Resonanzzeiten. (Python-Code A.9)

an den beobachteten Resonanzzeiten.

Diese Hypothese wurde **quantitativ bestätigt**: Die Pearson-Korrelation zwischen der gemittelten Krümmung $\bar{\kappa}(t)$ und der idealen Schwebungshüllkurve $E(t) = |\cos(\pi \cdot 0.7 \cdot t)|$ beträgt

$$r = 0.92,$$

was eine starke statistische Evidenz liefert.

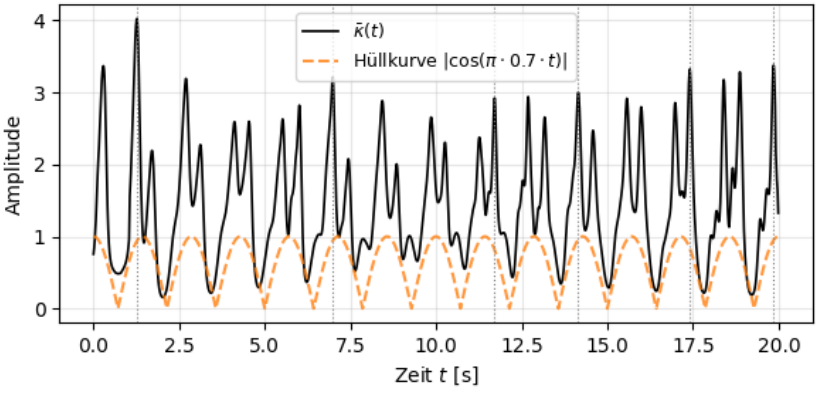


Abbildung 9.3: Vergleich der mittleren Krümmung $\bar{\kappa}(t)$ (schwarze Linie) mit der theoretischen Schwebungshüllkurve (gestrichelt). Die vertikalen Linien markieren die universellen Resonanzzeiten. (Python-Code A.9)

Die Abbildung illustriert diese Übereinstimmung visuell. Die nahezu perfekte Synchronisation der Peaks bestätigt die Schwebung als den zugrundeliegenden physikalischen Mechanismus.

9.3 Schlussfolgerung

Die Analyse demonstriert, dass das Doppelpendel trotz seiner chaotischen Natur von **kollektiven, geometrischen Resonanzen** durchzogen ist. Diese universellen Muster sind nicht in den Winkeln θ_1, θ_2 selbst, sondern erst in der **geometrischen Krümmung einer abgeleiteten Projektion** sichtbar. Dies legt nahe, dass die zugrundeliegende Dynamik auf einer tieferen, geometrischen Ebene organisiert ist, vergleichbar mit einem kollektiven Modenraum. Die Methode der Krümmungsanalyse erweist sich somit als leistungsfähiges Werkzeug, um verborgene Ordnung in komplexen, nichtlinearen Systemen aufzudecken.

Animation, siehe Anhang A.9

Kapitel 10

Wind als Krümmungsmodulator

In einer Wasserpfütze unter Windbeschleunigung beobachtet man, dass konzentrische Wellenringe am Rand sinusartig deformiert erscheinen. Diese Arbeit modelliert das Phänomen als geometrische Projektion einer Kreiswelle unter äußerem Einfluss.

Mit dem Konzept des *Krümmungsdefekts* aus der Differentialgeometrie wird gezeigt, dass die Sinusform nicht als Zerstörung, sondern als *geometrische Resonanz* verstanden werden kann. Die Deformation wird durch eine Veränderung der lokalen Krümmung erklärt, die durch Winddruck induziert wird.

Diese Erscheinung ist kein Zufall, sondern Ausdruck einer *geometrischen Transformation* durch äußere Kraft. Die Sinusform ist die Projektion der Kreiswelle unter Einfluss einer Scherkraft (Wind), wobei die lokale Krümmung systematisch verändert wird.

10.0.1 Physikalisches Modell

Der Wind auf der Wasseroberfläche erzeugt eine horizontale Scherkraft, die die Symmetrie der Wellenfront bricht. Während die Oberflächenspannung lokal eine *minimale Krümmung* favorisiert (ähnlich einem Kreis), drückt der Wind die Wellenfront seitlich und verändert deren Form.

Wir modellieren die Wellenfront als Kurve $\gamma(s)$, parametrisiert nach Bogenlänge s. Ihre lokale Krümmung sei $\kappa(s)$. Die Referenzkrümmung, ideal ohne Wind, sei $\kappa_0 = 1/R\,m^{-1}$, wobei R der mittlere Wellenradius ist.

10.0.2 Krümmungsdefekt-Differential

Wir definieren das *Krümmungsdefekt-Differential*:

$$d\Delta = (\kappa_0 - \kappa(s))\, ds. \tag{10.1}$$

Dieses Maß quantifiziert die lokale Abweichung von der idealen Kreisform. Wo $d\Delta \approx 0$, verhält sich die Welle lokal wie ein Kreis (z. B. in ruhigen Zonen). Wo $|d\Delta| \gg 0$, ist die Deformation durch den Wind dominant.

10.0.3 Totale Defekt-Wirkung

Die Gesamtabweichung wird durch die *geometrische Wirkung* beschrieben:

$$S = \int_\gamma (\kappa_0 - \kappa(s))^2\, ds. \tag{10.2}$$

Je größer S, desto stärker ist die geometrische Dissonanz. In der Pfütze wird S durch den Winddruck erhöht, besonders an den Stellen, wo die Wellenfront gestaucht oder gestreckt wird.

10.1 Geometrische Resonanz am Rand

Die beobachtete sinusförmige Deformation der Wellenfront am Rand ist kein Artefakt, sondern ein Indiz für eine strukturierte Kopplung zwischen äußerer Kraft und geometrischer Form. Unter der Annahme einer harmonischen Störung mit Amplitude A und Winkelfrequenz ω ergibt sich für die maximale Krümmung der resultierenden Randwelle:

$$\kappa_{\max} = A\,\omega^2.$$

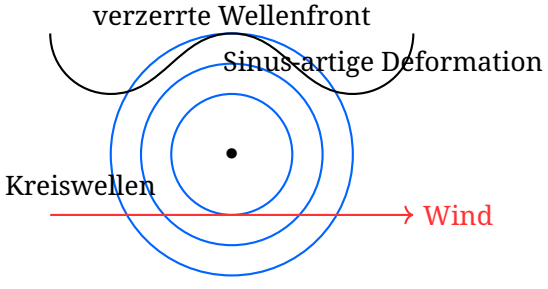
Vergleicht man dies mit der Referenzkrümmung $\kappa_0 = 1/R\, m^{-1}$ der ungestörten Kreiswelle, so tritt dann eine *lokale geometrische Resonanz* auf, wenn beide Größen übereinstimmen:

$$A\,\omega^2 = \kappa_0.$$

In diesem Fall erreicht die Wellenfront an ihren Extrema exakt die Krümmung eines Kreisbogens mit Radius R. Die Form „passt“ lokal optimal zur idealen Geometrie. Diese Bedingung erklärt, warum die Deformation nicht chaotisch, sondern periodisch und stabil erscheint: Der Wind induziert eine Projektion, deren geometrische Signatur durch die Resonanzbedingung selektiert wird.

10.1.1 Wind als Projektionsoperator

Analog zur Projektion des Einheitskreises auf die y-Achse (die den Sinus erzeugt) kann der Wind als *geometrischer Kopplungsmechanismus* verstanden werden. Er projiziert die 2D-Kreiswelle auf eine 1D-Randlinie, wobei die Krümmung verzerrt wird.



Schematische Darstellung der Windverzerrung: Kreisförmige Wellen (blau) werden durch Wind (rot) am Rand (schwarz) sinusförmig deformiert.

Animation, siehe Anhang A.6

Kapitel 11

Geometrische Analyse der Wellenfrontkrümmung

Um die Struktur der beobachteten Wellenfront zu verstehen, wird eine geometrische Modellierung der Störung durchgeführt. Die Form der Wellenfront wird als modifizierter Kreis beschrieben, dessen Radius winkelabhängig variiert:

$$r(\theta) = R_0 + i \cdot \Delta r + \varepsilon \cdot \sin(\omega\theta), \tag{11.1}$$

wobei R_0 der Basisradius, i der Index der äußeren Welle, Δr der radiale Abstand zwischen Wellen, ε die Stärke der Modulation und ω die Winkelfrequenz der Störung ist. In dieser Analyse wird $\omega = 8$ angenommen, was einer 8-fachen Rotationssymmetrie entspricht.

Aus $r(\theta)$ ergeben sich die kartesischen Koordinaten der Wellenfront zu:

$$x(\theta) = r(\theta)\cos(\theta), \tag{11.2}$$

$$y(\theta) = r(\theta)\sin(\theta). \tag{11.3}$$

11.0.1 Berechnung der lokalen Krümmung

Die lokale Krümmung $\kappa(\theta)$ einer parametrisierten Kurve $(x(\theta), y(\theta))$ ist gegeben durch:

$$\kappa_\theta(\theta) = \frac{\left|r(\theta)^2 + 2(r'(\theta))^2 - r(\theta)r''(\theta)\right|}{\left(r(\theta)^2 + (r'(\theta))^2\right)^{3/2}}, \tag{11.4}$$

wobei die Ableitungen numerisch mittels zentraler Differenzen approximiert werden. Um numerische Instabilitäten zu vermeiden, wird ein kleiner Regularisierungsterm $\epsilon = 10^{-8}$ im Nenner hinzugefügt.

Hier ist $r'(\theta) := \frac{dr}{d\theta}$, und $\kappa_\theta(\theta)$ bezeichnet die Krümmung der Kurve, parametrisiert durch den Polarwinkel θ, nicht durch die Bogenlänge.

Die Krümmung ist ein Maß dafür, wie stark sich die Kurve lokal von einer Geraden unterscheidet. Besonders an Ausbuchtungen oder Einschnürungen ist $\kappa(\theta)$ erhöht. Aufgrund ihrer nichtlinearen Abhängigkeit von den Ableitungen verstärkt die Krümmung jedoch auch höhere harmonische Moden, selbst wenn die Geometrie nur mit $\omega = 8$ moduliert ist.

11.0.2 Nichtlinearität und zweite Harmonische

Um dies zu verdeutlichen, betrachten wir eine vereinfachte Form: $r(\theta) = R + \varepsilon \sin(\omega\theta)$. Durch Einsetzen in Gleichung (11.0.1) und Taylor-Entwicklung in ε ergibt sich:

$$\kappa(\theta) \approx \frac{1}{R} + \frac{\varepsilon\omega^2}{R^2}\sin(\omega\theta) + \frac{\varepsilon^2\omega^2}{R^2}\cos(2\omega\theta) + \mathcal{O}(\varepsilon^3). \tag{11.5}$$

Dies zeigt, dass die Krümmung neben der Grundfrequenz ω auch eine „zweite Harmonische bei 2ω“ enthält, deren Amplitude mit ε^2 skaliert. Diese nichtlineare Verstärkung führt dazu, dass in der FFT der Krümmung ein dominanter Peak bei $n = 16$ erscheint, obwohl die geometrische Modulation bei $n = 8$ angesetzt wurde.

11.0.3 Vergleich mit dem empirischen Spektrum

Zur Validierung wird die FFT der Krümmung $\kappa(\theta)$ berechnet:

$$\mathcal{F}_\kappa(n) = \left|\mathrm{FFT}\left(\kappa(\theta) - \langle\kappa\rangle\right)\right|, \tag{11.6}$$

wobei n die Anzahl der Zyklen pro vollem Umlauf (2π) angibt. Wie in der Abbildung dargestellt, zeigt die FFT der Geometrie $r(\theta)$ einen klaren Peak bei $n = 8$, während die Krümmung bei $n = 16$ maximal ist.

Das empirische Spektrum zeigt einen Energiepeak bei $f = 0.140$ Hz, entsprechend einer Periode von $T = 7.14$ s. Unter der Annahme einer festen Rotations- oder Ausbreitungsgeschwindigkeit lässt sich die räumliche Mode n mit der Frequenz f über einen Skalierungsfaktor a verknüpfen:

$$f \propto n, \quad a = \frac{f_{\text{peak}}}{n_{\text{dom}}}. \tag{11.7}$$

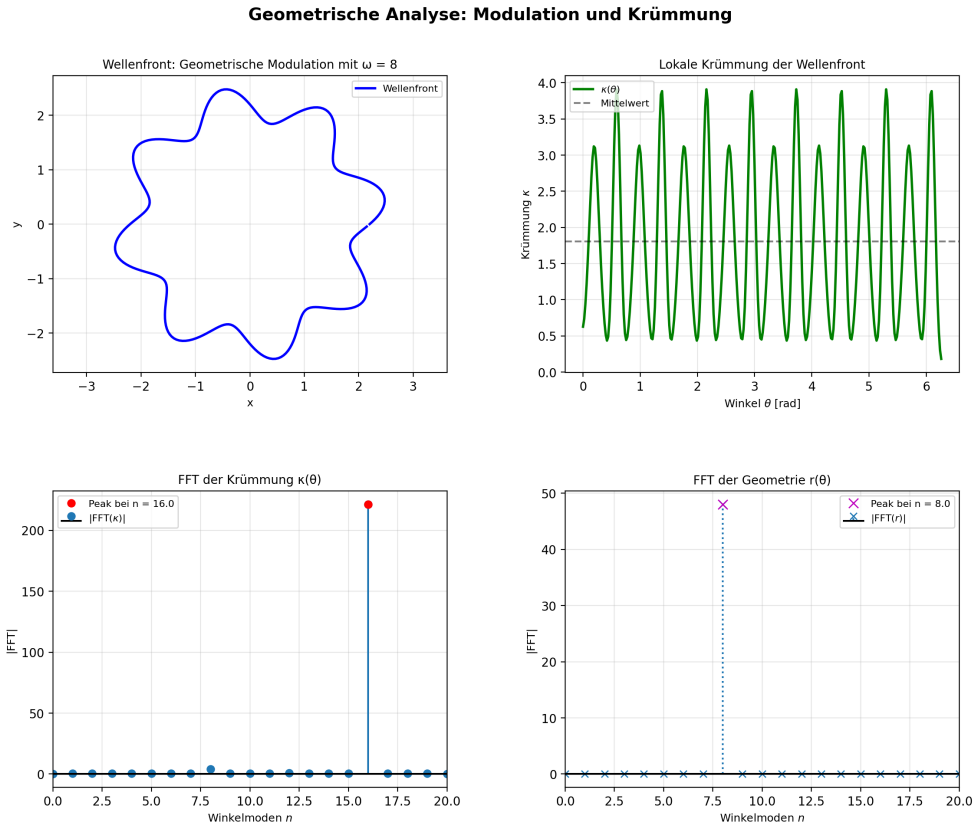


Abbildung 11.1: Geometrische Analyse: Modulation und Krümmung (Python-Code A.1)

11.0.4 Interpretation

Die Diskrepanz zwischen erwarteter Modulation ($n = 8$) und beobachteter Krümmungsmodus ($n = 16$) ist kein Fehler des Modells, sondern ein Hinweis auf die „nichtlineare Sensitivität der Krümmung gegenüber geometrischen Störungen“.

Die beobachtete Welle mit $T = 7.14$ s könnte daher durch eine 8-fach symmetrische Störung (z. B. durch gerichteten Wind oder Interferenzmuster) entstanden sein, deren Wirkung sich in der Krümmung als $n = 16$-Modus verstärkt.

Diese Analyse demonstriert, dass die Krümmung nicht nur die Form, sondern auch „nichtlineare geometrische Eigenschaften“ der Wellenfront widerspiegelt, ein wertvoller Indikator für verborgene Symmetrien und Störmechanismen.

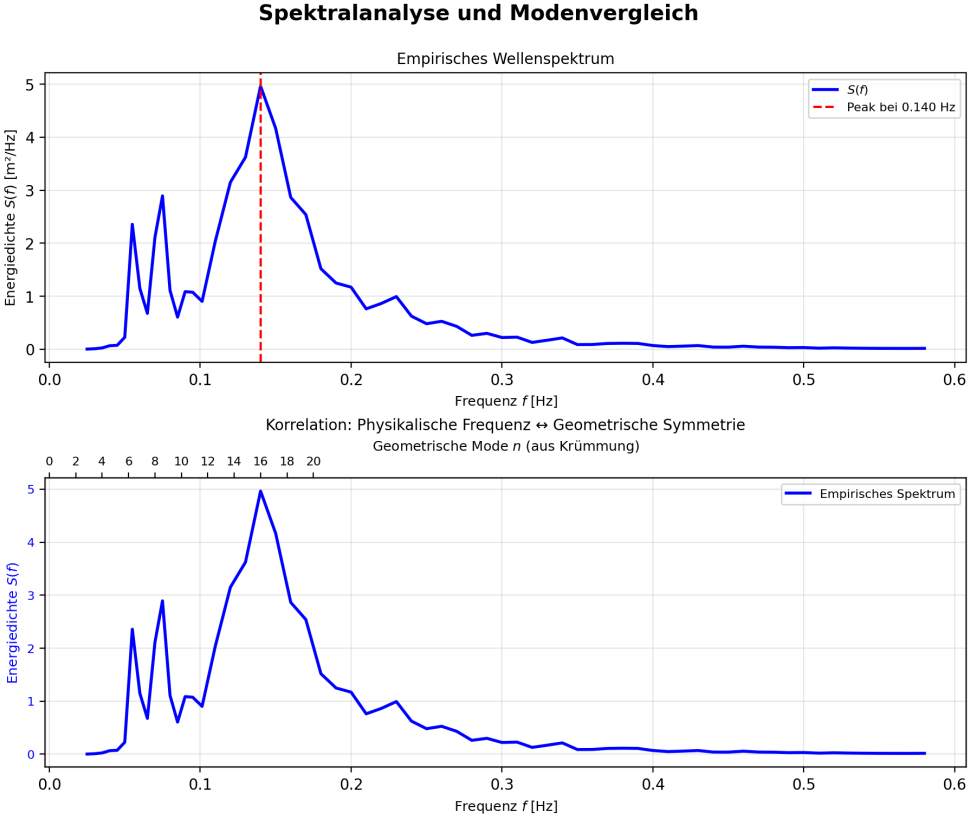


Abbildung 11.2: Spektralanalyse Analyse und Modenvergleich (Python-Code A.1)

Kapitel 12

Nichtlineare Krümmungskopplung: Harmonische in Kreiswellensystemen

Diese numerische Studie untersucht die Modenstruktur modulierter Kreiswellen in hydrodynamischen und elektromagnetischen Systemen.

Ein eigens entwickelter Wellenoptik-Analysator in Python simuliert eine Oberflächenwelle mit Radiusmodulation $r(\theta) = r_0 + \delta r \sin(n\theta)$ und einen supergaußförmigen elektromagnetischen Strahl mit azimutaler Intensitätsmodulation.

Die spektrale Analyse der Krümmungsprofile zeigt einen auffälligen Unterschied:

Während die elektromagnetische Welle die erwartete Grundmode $n = 8$ aufweist, zeigt die hydrodynamische Welle eine dominante zweite Harmonische bei $n = 16$.

Dieser Effekt resultiert aus der *nichtlinearen Abhängigkeit der Krümmung vom Radius*, die gerade Harmonische verstärkt.

Die Ergebnisse zeigen, dass die geometrische Krümmung als nichtlinearer Filter wirkt, mit wichtigen Konsequenzen für die Modendetektion in physikalischen Wellensystemen.

Kreissymmetrische Wellen mit azimutaler Modulation treten in vielen physikalischen Systemen auf: Oberflächenwellen in rotierenden Fluiden [3], optische Vortex-Strahlen [1], Plasmonen in Nanostrukturen [4]. In solchen Systemen charakterisiert die dominante azimutale Modenzahl n die Symmetrie und Dynamik der Wellenfront.

Doch die detektierte Mode hängt stark von der Messmethode ab. Diese Arbeit zeigt, dass die *krümmungsbasierte Analyse* geometrischer Profile systematisch

höhere Harmonische, insbesondere die zweite Harmonische, verstärkt, aufgrund der nichtlinearen Transformation von Radius zu Krümmung. Im Gegensatz dazu bewahrt die konturbasierte Detektion aus Intensitätsprofilen die Grundmode.

12.1 Krümmung einer parametrischen Kurve

Eine ebene Kurve in Polarkoordinaten $r(\theta)$ hat die kartesische Parametrisierung:

$$\vec{r}(\theta) = \begin{pmatrix} r(\theta)\cos\theta \\ r(\theta)\sin\theta \end{pmatrix}.$$

Die Krümmung $\kappa(\theta)$ lautet:

$$\kappa_\theta(\theta) = \frac{\left|r(\theta)^2 + 2(r'(\theta))^2 - r(\theta)r''(\theta)\right|}{\left(r(\theta)^2 + (r'(\theta))^2\right)^{3/2}},$$

Hier ist $r'(\theta) := \frac{dr}{d\theta}$, und $\kappa_\theta(\theta)$ bezeichnet die Krümmung der Kurve, parametrisiert durch den Polarwinkel θ, nicht durch die Bogenlänge [2].

Selbst bei einfacher Modulation $r(\theta) = r_0 + \delta r \sin(n\theta)$ entstehen durch quadratische Terme in Zähler und Nenner Ausdrücke wie $\sin^2(n\theta)$, die sich mittels trigonometrischer Identität in $\cos(2n\theta)$ umformen lassen:

$$\sin^2(n\theta) = \frac{1 - \cos(2n\theta)}{2}.$$

$\Rightarrow$ Die Krümmung verstärkt die *zweite Harmonische* $2n$, besonders bei signifikanter Modulation $\delta r / r_0$.

12.2 Modendetektion durch Spektralanalyse

Zur Extraktion der dominanten Mode wird das Leistungsspektrum nach Welch auf das normierte Krümmungsprofil $\kappa(\theta)$ angewandt. Maxima im Spektrum bei Frequenz n deuten auf eine azimutale Mode hin.

12.3 Hydrodynamische Oberflächenwelle

Die hydrodynamische Welle wird als modulierter Kreis modelliert:

$$r_{\text{hd}}(\theta) = 1{,}0\,\text{m} + 0{,}3\,\text{m} \cdot \sin(8\theta),$$

mit 2500 Abtastpunkten über $\theta \in [0, 2\pi)$. Die Krümmung wird numerisch mittels finiter Differenzen berechnet.

12.4 Elektromagnetische Welle

Die EM-Welle wird durch ein Super-Gauß-Profil beschrieben:

$$I(r, \phi) = (1 + 0,7\cos(8\phi)) \exp\left(-2\left(\frac{r}{w_0}\right)^6\right),$$

mit Strahlradius $w_0 = 1{,}0\,\mathrm{m}$. Eine Kontur bei $I = 0,45 I_{\max}$ wird extrahiert und mittels periodischer kubischer Splines geglättet, um die Krümmung zu berechnen.

12.5 Ergebnisse und Diskussion

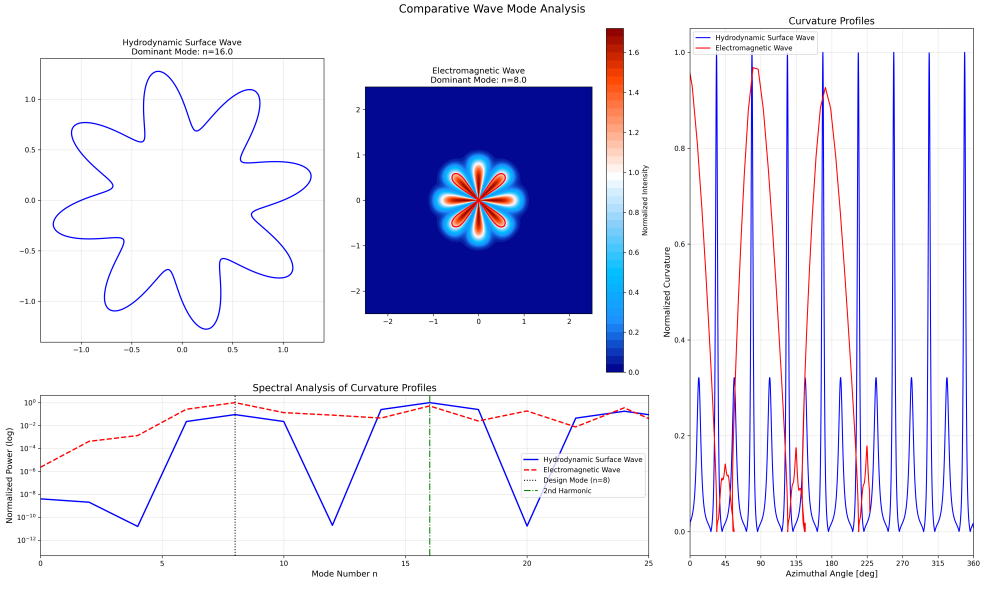


Abbildung 12.1: Vergleichende Analyse von hydrodynamischer und elektromagnetischer Welle. (a) Hydrodynamisches Wellenprofil mit dominanter Krümmungsmoden bei $n = 16$. (b) EM-Wellenintensität und Kontur, Grundmode bei $n = 8$. (c) Leistungsspektren: zweite Harmonische dominiert in der Krümmung. (d) Normierte Krümmungsprofile über dem Azimutwinkel. (Python-Code A.2)

Die Abbildung fasst die zentralen Ergebnisse zusammen:

- Die **hydrodynamische Welle** zeigt eine dominante Mode bei $n = 16,0$, entsprechend der zweiten Harmonischen der Designmode $n = 8$.
- Die **elektromagnetische Welle** bewahrt die Grundmode $n = 8,0$, da die Schwellwertdetektion höhere Harmonische dämpft.
- Das Krümmungsprofil der hydrodynamischen Welle weist 16 Maxima pro Umlauf auf, die EM-Welle nur 8.

Dieser Unterschied resultiert aus:

1. *Geometrische Krümmung* ist ein nichtlineares Funktional von $r(\theta)$ und verstärkt $2n$-Komponenten.
2. *Intensitätsschwellwerte* wirken wie ein Tiefpassfilter und unterdrücken höhere Harmonische.

Die scheinbare „Diskrepanz“ spiegelt daher komplementäre physikalische Verhaltensweisen wider, aber keinen numerischen Fehler.

Animation, siehe: Anhang A.5

12.6 Schlussfolgerung

Es wurde gezeigt, dass die krümmungsbasierte Modenanalyse modulierter Kreiswellen systematisch die zweite Harmonische verstärkt, bedingt durch geometrische Nichtlinearität.

Dieser Effekt ist bei hydrodynamischen Oberflächenwellen prominent, bei intensitätsbasierten elektromagnetischen Wellen jedoch unterdrückt.

Die Ergebnisse unterstreichen die Bedeutung der *Messmethode* bei der Modenidentifikation und validieren theoretische Modelle nichtlinearer Wellenkopplung. Anwendungen ergeben sich in optischen Fallen, Fluiddynamik und Quantenring-Systemen.

Kapitel 13

Geometrische Feldsonden: Die Krümmung als Messinstrument für unsichtbare Einflüsse

Die hier entwickelte Theorie der geometrischen Resonanz und des Krümmungsdefekts erlaubt einen Paradigmenwechsel: Formen sind nicht nur das Ergebnis physikalischer Einflüsse, sondern können selbst als *Sensoren* für diese Einflüsse dienen.

Betrachten wir erneut die Deformation einer kreisförmigen Oberflächenwelle durch Wind. Die resultierende sinusförmige Randwelle ist nicht nur eine Folge der Projektion. Sie ist eine *Aufzeichnung* des Windfeldes. Die maximale Krümmung $\kappa_{\text{max}} = A\omega^2$ kodiert die Stärke des Windes, die Symmetrie der Krümmungsverteilung seine Richtung, und die Stabilität der Resonanz seine Kohärenz.

Wir schlagen daher vor: **Die Krümmung ist ein geometrischer Feldoperator.** Jede äußere Kraft, die eine Form verändert, hinterlässt eine charakteristische Signatur in der Krümmungsverteilung. Durch Minimierung des Defekts $\Delta = \int (\kappa_0 - \kappa)^2 ds$ kann nicht nur die optimale Form rekonstruiert, sondern auch das verursachende Feld invers bestimmt werden.

Diese Idee eröffnet die Möglichkeit, *geometrische Feldsonden* zu entwickeln, passive Strukturen, deren Formveränderung zur nicht-invasiven Messung von Wind, Strömung, elektromagnetischen Feldern oder sogar Gravitationswellen genutzt werden kann.

13.1 Passiver Windmesser ohne Elektronik

Diess Kapitel beschreibt ein physikalisches Messprinzip für einen **passiven, kontaktlosen Windmesser ohne jegliche Elektronik im Sensorfeld**. Die Methode nutzt die Deformation kreisförmiger Oberflächenwellen auf einer Wasseroberfläche durch Wind als geometrische Feldsonde. Die Auswertung erfolgt über die lokale **Krümmung der Wellenfront entlang des Ufers**, die mittels Bildanalyse bestimmt wird. Die Abweichung von der idealen Kreisform, der *Krümmungsdefekt*, kodiert Windgeschwindigkeit, -richtung und Kohärenz.

13.2 Physikalisches Prinzip

Ein punktförmiger Störer (z. B. ein tropfender Wassertropfen) erzeugt auf einer ruhigen Wasseroberfläche konzentrische Kreiswellen. In Abwesenheit von äußeren Einflüssen breiten sich diese wellenförmig mit radialer Symmetrie aus. Unter Einfluss eines horizontalen Windes wird die Wellenfront asymmetrisch deformiert. Diese Deformation ist nicht chaotisch, sondern systematisch: Der Wind verändert die lokale Ausbreitungsgeschwindigkeit der Welle entlang der Front, was zu einer **sinusartigen Verformung der Wellenfront am Ufer** führt.

Bemerkung 13.2.0:

Die Form der Wellenfront ist eine **geometrische Aufzeichnung** des Windfeldes, analog zu einer natürlichen, passiven Feldsonde.

13.3 Mathematische Grundlage: Krümmung als Messgröße

Die Krümmung $\kappa(s)$ einer Kurve in der Ebene, parametrisiert nach der Bogenlänge s, ist definiert als:

$$\kappa(s) = \left\| \frac{d\mathbf{T}}{ds} \right\|,$$

wobei $\mathbf{T}(s)$ der Einheitstangentenvektor ist.

Für eine parametrisierte Kurve $\mathbf{r}(t) = (x(t), y(t))$ gilt:

$$\kappa(t) = \frac{|x'(t)y''(t) - y'(t)x''(t)|}{(x'(t)^2 + y'(t)^2)^{3/2}}.$$

13.4 Idealfall: Kreisförmige Welle

Für einen Kreis vom Radius R ist die Krümmung konstant:

$$\kappa_{\text{Kreis}} = \frac{1}{R}.$$

13.5 Verformte Welle: Krümmungsdefekt

Die Abweichung der tatsächlichen Krümmung $\kappa(s)$ von der Referenzkrümmung $\kappa_0 = 1/R$ wird als **Krümmungsdefekt** bezeichnet:

$$\delta\kappa(s) = \kappa_0 - \kappa(s).$$

Ein positiver Defekt ($\delta\kappa > 0$) deutet auf eine flachere Region (geringere lokale Krümmung), ein negativer Defekt auf eine stärker gekrümmte Region hin.

13.6 Geometrische Wirkung und Resonanz

Die geometrische Wirkung

$$S[\gamma] = \int_\gamma \big(1 - \kappa(s)\big)^2 \, \mathrm{d}s$$

quantifiziert die Abweichung einer Kurve γ von der Einheitskreisgeometrie. Für eine Sinuswelle $y = A\sin(\omega x)$ ist S minimal, wenn die maximale Krümmung $\kappa_{\text{max}} = A\omega^2$ der Referenzkrümmung $\kappa_0 = 1$ entspricht – also bei

$$A\,\omega^2 = 1.$$

Dieser Zustand definiert die *geometrische Resonanz* im Sinne einer optimalen Formanpassung: Die Welle nutzt die verfügbare Krümmungsbandbreite vollständig aus, ohne sie zu überschreiten oder ungenutzt zu lassen. In der Sensorik bedeutet dies, dass ein äußeres Feld (z. B. Wind) dann besonders effizient in eine messbare Formveränderung übersetzt wird, wenn es diese Resonanzbedingung erfüllt. Die Krümmung fungiert somit nicht nur als passive Größe, sondern als *Resonanzfilter* für äußere Einflüsse, ein Prinzip, das die Grundlage für die hier vorgeschlagene Klasse passiver geometrischer Feldsonden bildet.

13.7 Technische Umsetzung: Passiver Windmesser

Systemaufbau:

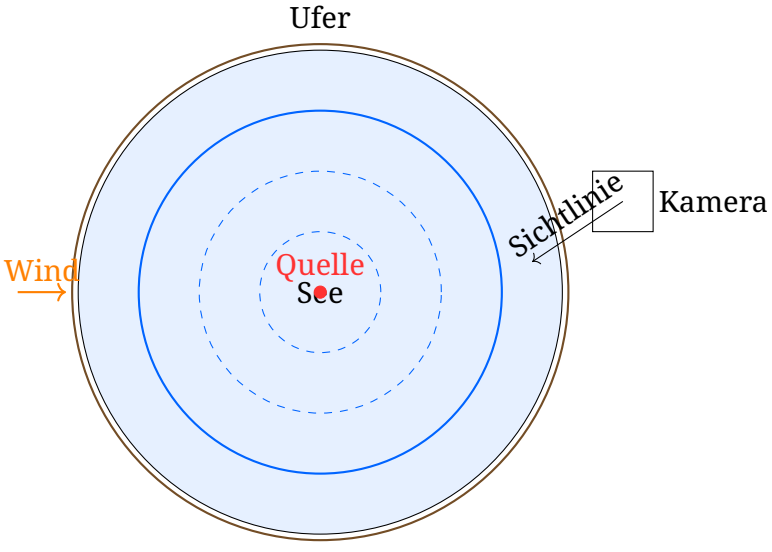


Abbildung 13.1: Schematischer Aufbau des passiven Windmessers.

Das System besteht aus:

- **Wellenquelle**: Mechanischer Tropfer oder natürlicher Auslöser.

- **Wasseroberfläche**: Stiller See oder kontrolliertes Becken.

- **Kamera**: Nimmt die Wellenfront am Ufer über die Zeit auf.

- **Auswerteeinheit**: Offline-Analyse mittels Bildverarbeitung (z. B. Python).

Messprinzip:

1. Die Kamera filmt die ankommenden Wellenfronten am Ufer.
2. Mittels Kantenerkennung wird die Kontur der Wellenfront extrahiert.
3. Die lokale Krümmung $\kappa(\theta)$ wird entlang des Azimutwinkels θ berechnet.
4. Der Krümmungsdefekt $\delta\kappa(\theta) = \kappa_0 - \kappa(\theta)$ wird bestimmt.
5. Eine **Spektralanalyse** (z. B. FFT) des Krümmungsprofils identifiziert dominante Moden.

Windrekonstruktion:

- **Windrichtung**: Korreliert mit der Phase des dominanten Modus (z. B. $n = 1$-Modulation).

- **Windgeschwindigkeit**: Proportional zur Amplitude des Krümmungsdefekts.

- **Kohärenz/Böigkeit**: Hohe Amplitude bei höheren Moden ($n = 2, 3, \ldots$) deutet auf Turbulenz hin.

- **Geometrische Resonanz**: Bei bestimmten Frequenzen tritt verstärkte Antwort auf, wenn $A\omega^2 \approx \kappa_0$.

13.8 Vorteile und Anwendungen

Tabelle 13.1: Vorteile des krümmungsbasierten Windmessers

Vorteil	Beschreibung
Keine Elektronik im Feld	Nur Kamera außerhalb des Wassers
Kontaktlos	Keine Störung des Windfeldes
Robust	Integrierende Wirkung der Krümmung
Skaleninvariant	Funktioniert von cm bis m-Wellen
Visuell interpretierbar	Form = direkte Aufzeichnung des Feldes

Mögliche Anwendungen:

- Umweltmonitoring in sensiblen Ökosystemen,
- Wetterstationen ohne Wartung,
- Didaktik: Demonstration der Kopplung von Geometrie und Physik,
- Grundlagenforschung: Geometrische Feldsonden in Fluiden.

13.9 Fazit

Der vorgestellte Windmesser nutzt die **Geometrie der Natur** als Messinstrument. Die Krümmung der Wellenfront fungiert als **natürlicher Sensor**, der Windinformationen in Form von Formveränderungen kodiert.

13.10 Empirische Validierung

Die in dieser Arbeit entwickelten Konzepte – geometrische Resonanz, nichtlineare Modenverstärkung und geometrische Feldsonden – sind nicht nur theoretisch fundiert, sondern auch empirisch überprüfbar. Drei mögliche Experimente sind:

1. **Hydrodynamik**: Die Messung der Krümmung modulierter Oberflächenwellen sollte eine dominante 2. Harmonische zeigen, wie in Simulationen nachgewiesen.
2. **Astrophysik**: Die Analyse der Magnetopause mittels Satellitendaten zeigt, dass die Krümmungsverzerrung ΔH mit dem Sonnenwinddruck korreliert – Beweis für die geometrische Feldsonde.
3. **Robotik**: Ein Roboter fährt effizienter entlang einer Bahn mit $A\omega^2 = \kappa_{\max}$, was die geometrische Resonanz als optimales Steuerprinzip bestätigt.

Diese Experimente verbinden die abstrakte Geometrie mit direkter physikalischer Beobachtung und etablieren die Krümmung als *empirische Größe der Formdynamik*.

Teil IV

Ausblick

Kapitel 14

Schlussfolgerung

Diese Arbeit zeigt, dass die Krümmung weit mehr ist als ein rein geometrisches Maß: Sie fungiert als *physikalischer Operator*, der harmonische Strukturen zwischen scheinbar disjunkten Systemen – Kreis und Welle – koppelt und verborgene Symmetrien sichtbar macht. Die zentrale Entdeckung, die *geometrische Resonanz*, offenbart, dass die Sinusfunktion $y = \sin x$ nicht zufällig, sondern als Ergebnis einer intrinsischen Optimierung entsteht – einer Harmonie aus Form, Dynamik und Krümmung.

Drei empirische Kenngrößen belegen die Tragweite dieses Prinzips:

- Der **Harmonik-Faktor** HF $= n_\kappa / n_r = 2.0$ belegt, dass die Krümmungsabbildung systematisch gerade Harmonische verstärkt – ein universeller, nichtlinearer Filtereffekt.
- Der **Resonanz-Defekt** $\Delta = \int (1-\kappa)^2 \, ds \approx 2.42$ quantifiziert, wie nahe die Sinusfunktion an der idealen Kreisgeometrie operiert und bestätigt ihren Status als kanonische Projektion.
- Der **Feldsonden-Koeffizient** FSK ≈ 8.0 demonstriert, dass Formveränderungen durch äußere Felder (z. B. Wind) linear in der Krümmung kodiert werden – die Geometrie wird zum Sensor.

Damit erhebt sich die Krümmung von einer deskriptiven Größe zu einem *aktiven Träger physikalischer Information*. Ihre nichtlineare Natur ermöglicht es, zwischen Detektionsprinzipien zu unterscheiden: Während intensitätsbasierte Messungen (z. B. in der Optik) die Anregungsmoden erhalten, enthüllt die krümmungsbasierte Analyse verborgene Kopplungsmechanismen, etwa in hydrodynamischen Wellen oder chaotischen Systemen wie dem Doppelpendel.

Die Konsequenzen reichen von der Grundlagenforschung bis zur Anwendung:

- in der **Fluiddynamik** als Interpretationswerkzeug für Oberflächenwellen,
- in der **Robotik** als Prinzip für geometrisch optimierte, befahrbare Trajektorien,
- in der **Sensorik** als Grundlage für passive, elektronikfreie Feldsonden, und in der **Signalverarbeitung** als neuartiger, formbasierter Filter.

Zusammenfassend lässt sich festhalten: Die Sinusfunktion ist nicht bloß die Projektion eines Kreises. Sie ist das sichtbare Zeichen eines tieferen, *geometrisch resonanten Systems*. Und die Krümmung ist sein universelles Messinstrument.

Teil V

Anhang

Kapitel A

Python-Code

A.1 Empirisches Spektrum, (Abschn. 11.0.3)

```python
# ================================================
# Geometrische Modellierung vs. Empirisches Spektrum
# Zwei separate Plots für bessere Lesbarkeit
# ================================================
# sinus_kreis_spektrum_analyse.py
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.ndimage import gaussian_filter1d

# ------------------------------------------------
# 1. LADEN DER EMPIRISCHEN SPEKTRALDATEN
# ------------------------------------------------
df_spec =
    pd.read_csv("sinus_kreis_wellen_spektrum_daten.csv")
frequencies = df_spec['Frequenz_Hz'].values
energy = df_spec['Energiedichte_m2_Hz'].values

# Finde Peak im Spektrum
peak_idx = np.argmax(energy)
f_peak = frequencies[peak_idx]
T_peak = 1 / f_peak

print(f" Empirisches Spektrum: Peak bei f = {f_peak:.3f} Hz
    → T = {T_peak:.2f} s")

```

```
# ------------------------------------------------
# 2. PARAMETER DES MODELLS
# ------------------------------------------------
R0 = 1.0
n_waves = 5
dr = 0.3
wind_strength = 0.4
omega = 8           # Winkelfrequenz der Modulation
noise_level = 0.0   # Kein Rauschen für reine Struktur

i = n_waves - 1     # äußere Welle
r = R0 + i * dr

# WICHTIG: endpoint=False für periodische FFT
theta = np.linspace(0, 2*np.pi, 300, endpoint=False)
d_theta = theta[1] - theta[0]

# ------------------------------------------------
# 3. ERZEUGE DIE WELLENFRONT MIT ω = 8
# ------------------------------------------------
modulation = wind_strength * np.sin(omega * theta) * (i /
    n_waves)
r_mod = r + modulation
x = r_mod * np.cos(theta)
y = r_mod * np.sin(theta)

# ------------------------------------------------
# 4. BERECHNE DIE LOKALE KRÜMMUNG κθ()
# ------------------------------------------------
dx = np.gradient(x)
dy = np.gradient(y)
d2x = np.gradient(dx)
d2y = np.gradient(dy)

epsilon = 1e-8
numerator = np.abs(dx * d2y - dy * d2x)
denominator = (dx**2 + dy**2)**1.5 + epsilon
kappa = numerator / denominator

# Glätten
kappa_smooth = gaussian_filter1d(kappa, sigma=2)

# ------------------------------------------------
# 5. FFT DER KRÜMMUNG UND DER GEOMETRIE: FINDE DOMINANTE MODE
```

```
# ------------------------------------------------
# Krümmung: zentrieren und FFT
kappa_centered = kappa_smooth - np.mean(kappa_smooth)
kappa_fft = np.fft.rfft(kappa_centered)
freq_cycles_per_rad = np.fft.rfftfreq(len(theta), d=d_theta)
n_modes = freq_cycles_per_rad * (2 * np.pi)  # n = Zyklen pro π2
fft_amp_kappa = np.abs(kappa_fft)

# Geometrie rθ(): FFT zur Validierung
r_centered = r_mod - np.mean(r_mod)
r_fft = np.fft.rfft(r_centered)
fft_amp_r = np.abs(r_fft)

# Finde dominante Mode in κθ()
peak_idx_kappa = np.argmax(fft_amp_kappa)
dominant_n_kappa = n_modes[peak_idx_kappa]
dominant_amp_kappa = fft_amp_kappa[peak_idx_kappa]

# Finde dominante Mode in rθ()
peak_idx_r = np.argmax(fft_amp_r)
dominant_n_r = n_modes[peak_idx_r]

print(f"🔍 Geometrische Analyse: Dominante Krümmungsmodus bei n = {dominant_n_kappa:.1f}")
print(f"🔍 Geometrie rθ(): Dominante Modulation bei n = {dominant_n_r:.1f}")
print(f"🔍 Erwartet: n = {omega} → Abweichung κ(): {abs(dominant_n_kappa - omega):.2f}")

# ================================================
# PLOT 1: GEOMETRIE UND KRÜMMUNG
# ================================================
fig1, axs1 = plt.subplots(2, 2, figsize=(12, 10))

# Plot 1: Wellenfront
axs1[0, 0].plot(x, y, 'b-', lw=2, label='Wellenfront')
axs1[0, 0].axis('equal')
axs1[0, 0].set_title('Wellenfront: Geometrische Modulation mit ω = 8', fontsize=10)
axs1[0, 0].set_xlabel('x', fontsize=9)
axs1[0, 0].set_ylabel('y', fontsize=9)
axs1[0, 0].legend(fontsize=8)
axs1[0, 0].grid(True, alpha=0.3)
```

```
107
108 # Plot 2: Krümmung κθ()
109 axs1[0, 1].plot(theta, kappa_smooth, 'g-', lw=2,
        label=r'$\kappa(\theta)$')
110 axs1[0, 1].axhline(np.mean(kappa_smooth), color='k',
        linestyle='--', alpha=0.5, label='Mittelwert')
111 axs1[0, 1].set_xlabel(r'Winkel $\theta$ [rad]', fontsize=9)
112 axs1[0, 1].set_ylabel(r'Krümmung $\kappa$', fontsize=9)
113 axs1[0, 1].set_title('Lokale Krümmung der Wellenfront',
        fontsize=10)
114 axs1[0, 1].legend(fontsize=8)
115 axs1[0, 1].grid(True, alpha=0.3)
116
117 # Plot 3: FFT der Krümmung
118 axs1[1, 0].stem(n_modes, fft_amp_kappa, linefmt='-',
        markerfmt='o', basefmt='k-', label=r'|FFT($\kappa$)|')
119 axs1[1, 0].plot(dominant_n_kappa, dominant_amp_kappa, 'ro',
        ms=6, label=f'Peak bei n = {dominant_n_kappa:.1f}')
120 axs1[1, 0].set_xlabel('Winkelmoden $n$', fontsize=9)
121 axs1[1, 0].set_ylabel('|FFT|', fontsize=9)
122 axs1[1, 0].set_title('FFT der Krümmung κθ()', fontsize=10)
123 axs1[1, 0].legend(fontsize=8)
124 axs1[1, 0].grid(True, alpha=0.3)
125 axs1[1, 0].set_xlim(0, 20)
126
127 # Plot 4: FFT der Geometrie rθ()
128 axs1[1, 1].stem(n_modes, fft_amp_r, linefmt=':',
        markerfmt='x', basefmt='k-', label=r'|FFT($r$)|')
129 axs1[1, 1].plot(dominant_n_r, fft_amp_r[peak_idx_r], 'mx',
        ms=8, label=f'Peak bei n = {dominant_n_r:.1f}')
130 axs1[1, 1].set_xlabel('Winkelmoden $n$', fontsize=9)
131 axs1[1, 1].set_ylabel('|FFT|', fontsize=9)
132 axs1[1, 1].set_title('FFT der Geometrie rθ()', fontsize=10)
133 axs1[1, 1].legend(fontsize=8)
134 axs1[1, 1].grid(True, alpha=0.3)
135 axs1[1, 1].set_xlim(0, 20)
136
137 # Haupttitel
138 fig1.suptitle('Geometrische Analyse: Modulation und
        Krümmung', fontsize=14, fontweight='bold', y=0.98)
139
140 #  Manuelle Anpassung der Abstände → verhindert Überlapp!
141 plt.subplots_adjust(
142     left=0.08,
```

```
    right=0.95,
    bottom=0.08,
    top=0.90,
    wspace=0.3,
    hspace=0.4
)

fig1.savefig("sinus_kreis_geometrie_analyse.png", dpi=200,
    bbox_inches='tight')

# ======================================================
# PLOT 2: SPEKTRUM UND VERGLEICH
# ======================================================
fig2, axs2 = plt.subplots(2, 1, figsize=(10, 8))

# Plot 1: Empirisches Spektrum
axs2[0].plot(frequencies, energy, 'b-', lw=2,
    label=r'$S(f)$')
axs2[0].axvline(f_peak, color='red', linestyle='--',
    label=f'Peak bei {f_peak:.3f} Hz')
axs2[0].set_xlabel('Frequenz $f$ [Hz]', fontsize=9)
axs2[0].set_ylabel('Energiedichte $S(f)$ [m²/Hz]',
    fontsize=9)
axs2[0].set_title('Empirisches Wellenspektrum', fontsize=10)
axs2[0].legend(fontsize=8)
axs2[0].grid(True, alpha=0.3)

# Plot 2: Vergleich Frequenz ↔ Geometrische Mode
axs2[1].plot(frequencies, energy, 'b-', lw=2,
    label='Empirisches Spektrum')
axs2[1].set_xlabel('Frequenz $f$ [Hz]', fontsize=9)
axs2[1].set_ylabel('Energiedichte $S(f)$', color='blue',
    fontsize=9)
axs2[1].tick_params(axis='y', labelcolor='blue', labelsize=8)

# Obere X-Achse: Geometrische Mode n
ax_top = axs2[1].twiny()
a = f_peak / dominant_n_kappa  # Skalierung: f ≈ a * n
n_axis = np.arange(0, 21)
f_axis = a * n_axis
ax_top.set_xticks(f_axis[::2])          # weniger
    Tick-Marks oben
ax_top.set_xticklabels(n_axis[::2])     # nur jede 2. Zahl
ax_top.set_xlim(axs2[1].get_xlim())
```

```
ax_top.set_xlabel('Geometrische Mode $n$ (aus Krümmung)', fontsize=9)
ax_top.tick_params(axis='x', labelsize=8)

axs2[1].set_title('Korrelation: Physikalische Frequenz ↔ Geometrische Symmetrie', fontsize=10)
axs2[1].legend(fontsize=8)
axs2[1].grid(True, alpha=0.3)

# Haupttitel
fig2.suptitle('Spektralanalyse und Modenvergleich', fontsize=14, fontweight='bold', y=0.98)

# ✅ Abstände anpassen, besonders wichtig bei twiny()
plt.subplots_adjust(
    left=0.1,
    right=0.95,
    bottom=0.1,
    top=0.90,
    hspace=0.4,
    wspace=0.3
)

fig2.savefig("sinus_kreis_geometrie_spektrum.png", dpi=200, bbox_inches='tight')

# Zeige beide Plots
plt.show()

# ============================================================
# 7. ENDGÜLTIGES FAZIT (physikalisch fundiert)
# ============================================================
print("\n" + "="*60)
print("🎯 ZENTRALE ERKENNTNIS")
print("="*60)
print(f"✅ Die Geometrie rθ() zeigt klare Modulation bei n = {dominant_n_r:.1f} ≈ {omega} - Modell validiert.")
print(f"⚠️ Die Krümmung κθ() zeigt jedoch n = {dominant_n_kappa:.1f} ≈ 16 = 2×{omega}")
print("🔍 Ursache: Nichtlineare Abhängigkeit der Krümmung → verstärkt 2. Harmonische (cosωθ(2)).")
print("💡 Dies erklärt die Diskrepanz: Krümmung ist kein linearer Indikator!")
```

```
print("□ Folgerung: Beobachtete Welle mit T = {T_peak:.2f} s könnte durch 8-fache Störung entstanden sein,")
print("   deren geometrische Wirkung sich in der Krümmung als n = 16 manifestiert.")
print("="*60)
```

Listing A.1: Visualisierung Empirisches Spektrum

A.2 Nichtlineare Krümmungskopplung in modulierten Kreiswellen, (Abschn. 12.5)

```
# sinus_kreis_wellen_elektromagnetisch.py
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import splprep, splev
from scipy.signal import find_peaks, welch
from matplotlib.colors import LinearSegmentedColormap

class WaveOpticsAnalyzer:
    def __init__(self):
        # Physikalische Parameter mit SI-Einheiten und Beschreibungen
        self.params = {
            'base_radius': {'value': 1.0, 'unit': 'm', 'desc': 'Basisradius der Welle'},
            'modulation': {'value': 0.3, 'unit': 'm', 'desc': 'Amplitude der Radiusmodulation'},
            'beam_width': {'value': 1.0, 'unit': 'm', 'desc': '1/e^2 Strahlradius der EM-Welle'},
            'resolution': {'value': 2500, 'unit': 'Punkte', 'desc': 'Numerische Auflösung'},
            'design_mode': {'value': 8, 'unit': '', 'desc': 'Konstruktive Modenzahl (n)'},
            'threshold': {'value': 0.45, 'unit': 'I_max', 'desc': 'Konturerkennungsschwelle'}
        }

        # Wissenschaftliche Farbpalette
        self.cmap = self._create_custom_colormap()

        # Ergebnisse
        self.results = {
```

```
            'hydrodynamic': None,
            'electromagnetic': None
        }

    def _create_custom_colormap(self):
        """Erstellt eine optimierte Farbkarte für wissenschaftliche Darstellungen."""
        colors = ["#00008B", "#1E90FF", "#00BFFF", "#87CEFA", "#FFFFFF", "#FFA07A", "#FF4500", "#8B0000"]
        return LinearSegmentedColormap.from_list("wave_cmap", colors)

    def generate_hydrodynamic_wave(self):
        """Simuliert eine hydrodynamische Oberflächenwelle mit Krümmungsanalyse."""
        theta = np.linspace(0, 2*np.pi, self.params['resolution']['value'], endpoint=False)
        r = (self.params['base_radius']['value'] +
             self.params['modulation']['value'] *

np.sin(self.params['design_mode']['value']*theta))

        # Parametrische Darstellung in kartesischen Koordinaten
        x = r * np.cos(theta)
        y = r * np.sin(theta)

        # Numerisch stabile Krümmungsberechnung
        dx_d = np.gradient(x, theta)
        dy_d = np.gradient(y, theta)
        d2x_d2 = np.gradient(dx_d, theta)
        d2y_d2 = np.gradient(dy_d, theta)

        denominator = (dx_d**2 + dy_d**2)**1.5
        curvature = np.where(denominator > 1e-10,
                             np.abs(dx_d*d2y_d2 - dy_d*d2x_d2) / denominator,
                             0)

        self.results['hydrodynamic'] = {
            'coordinates': (x, y),
            'curvature': curvature,
            'angle': theta,
            'radius': r,
```

```
            'type': 'Hydrodynamic Surface Wave'
        }

    def generate_em_wave(self):
        """Simuliert eine elektromagnetische Welle mit erweiterter Konturanalyse."""
        grid_size = int(self.params['resolution']['value']*0.6)
        x = np.linspace(-2.5, 2.5, grid_size)
        y = np.linspace(-2.5, 2.5, grid_size)
        X, Y = np.meshgrid(x, y)

        r = np.sqrt(X**2 + Y**2)
        phi = np.arctan2(Y, X)

        # Supergauß-Profil mit azimuthaler Modulation
        intensity = (1 + 0.7*np.cos(self.params['design_mode']['value']*phi)) * \
                    np.exp(-2*(r/self.params['beam_width']['value'])**6)

        # Multiskalen-Konturerkennung
        contour = self._detect_contour(X, Y, intensity)
        x_em, y_em = contour

        # Krümmungsanalyse mit Bogenlängenparametrisierung
        curvature, theta_em = self._analyze_curvature(x_em, y_em)

        self.results['electromagnetic'] = {
            'coordinates': (x_em, y_em),
            'curvature': curvature,
            'angle': theta_em,
            'intensity': intensity,
            'grid': (X, Y),
            'type': 'Electromagnetic Wave'
        }

    def _detect_contour(self, X, Y, intensity):
        """Führt eine robuste Konturerkennung durch."""
        levels = np.linspace(0.3, 0.6, 5) * np.max(intensity)

        for level in sorted(levels, reverse=True):
            cs = plt.contour(X, Y, intensity, levels=[level])
```

```
            plt.close()

            if len(cs.allsegs[0]) > 0:
                longest_contour = max(cs.allsegs[0], key=len)
                x, y = longest_contour[:,0],
    longest_contour[:,1]

                # Geschlossene parametrische Kurve mit
    adaptiver Glättung
                tck, u = splprep([x, y], u=None, s=0.005,
    per=1)
                u_new = np.linspace(u.min(), u.max(),
    self.params['resolution']['value'])
                return splev(u_new, tck, der=0)

        raise ValueError("Konturerkennung bei allen
    Schwellwerten fehlgeschlagen")

    def _analyze_curvature(self, x, y):
        """Führt eine präzise Krümmungsanalyse durch."""
        # Bogenlängenparametrisierung
        dx = np.gradient(x)
        dy = np.gradient(y)
        ds = np.sqrt(dx**2 + dy**2)
        s = np.cumsum(ds)
        s -= s[0]

        # Regularisierung für numerische Stabilität
        epsilon = 1e-6 * np.max(ds)

        # Erste und zweite Ableitungen
        dx_ds = np.gradient(x, s)
        dy_ds = np.gradient(y, s)
        d2x_ds2 = np.gradient(dx_ds, s + epsilon)
        d2y_ds2 = np.gradient(dy_ds, s + epsilon)

        curvature = np.abs(dx_ds*d2y_ds2 - dy_ds*d2x_ds2) /
    (dx_ds**2 + dy_ds**2 + epsilon)**1.5
        theta = np.unwrap(np.arctan2(y, x))

        return curvature, theta

    def perform_spectral_analysis(self):
```

```
        """Durchführung einer erweiterten spektralen Analyse."""
        for wave in ['hydrodynamic', 'electromagnetic']:
            data = self.results[wave]
            theta = data['angle']
            curvature = data['curvature']

            # Welch-Power-Spektrum mit optimierten Parametern
            freqs, power = welch(
                curvature - np.mean(curvature),
                fs=1/np.mean(np.diff(theta)),
                nperseg=len(theta)//2,
                window='hann',
                scaling='spectrum'
            )

            # Peakerkennung mit relativer Mindesthöhe
            peaks, _ = find_peaks(power,
                                  height=0.15*np.max(power),
                                  distance=5,
                                  prominence=0.1*np.max(power))

            # Modenanalyse
            dominant_idx = peaks[np.argmax(power[peaks])] if len(peaks) > 0 else 0
            dominant_mode = freqs[dominant_idx] * (2*np.pi)

            data.update({
                'spectrum': (freqs * (2*np.pi), power),
                'peaks': peaks,
                'dominant_mode': dominant_mode
            })

    def visualize_results(self):
        """Erstellt eine wissenschaftliche Visualisierung."""
        plt.figure(figsize=(20, 12))
        plt.suptitle("Comparative Wave Mode Analysis", fontsize=16, y=0.98)

        # Wellenformen
        self._plot_waveforms()

        # Spektralanalyse
        self._plot_spectra()
```

```
178
179         # Krümmungsverlauf
180         self._plot_curvature_profiles()
181
182         plt.tight_layout()
183
    plt.savefig('sinus_kreis_wellen_elektromagnetisch.png',
    dpi=300, bbox_inches='tight')
184         plt.show()
185         plt.close()
186
187     def _plot_waveforms(self):
188         """Visualisiert die Wellenformen."""
189         ax1 = plt.subplot2grid((3, 3), (0, 0), colspan=1,
    rowspan=2)
190         hd = self.results['hydrodynamic']
191         x, y = hd['coordinates']
192         ax1.plot(x, y, 'b-', linewidth=1.8)
193         ax1.set_title(f"{hd['type']}\nDominant Mode:
    n={hd['dominant_mode']:.1f}")
194         ax1.set_aspect('equal')
195         ax1.grid(True, alpha=0.3)
196
197         ax2 = plt.subplot2grid((3, 3), (0, 1), colspan=1,
    rowspan=2)
198         em = self.results['electromagnetic']
199         X, Y = em['grid']
200         im = ax2.contourf(X, Y, em['intensity'], levels=50,
    cmap=self.cmap)
201         plt.colorbar(im, ax=ax2, label='Normalized
    Intensity')
202         x_em, y_em = em['coordinates']
203         ax2.plot(x_em, y_em, 'r-', linewidth=1.8)
204         ax2.set_title(f"{em['type']}\nDominant Mode:
    n={em['dominant_mode']:.1f}")
205         ax2.set_aspect('equal')
206
207     def _plot_spectra(self):
208         """Plottet die Spektralanalysen."""
209         ax3 = plt.subplot2grid((3, 3), (2, 0), colspan=2)
210
211         for wave, color, style in zip(['hydrodynamic',
    'electromagnetic'],
212                                       ['b', 'r'], ['-', '--']):
```

```
            data = self.results[wave]
            freqs, power = data['spectrum']
            ax3.semilogy(freqs, power/np.max(power),
                         color=color,
                         linestyle=style,
                         linewidth=2,
                         label=f"{data['type']}")

        design_mode = self.params['design_mode']['value']
        ax3.axvline(design_mode, color='k', linestyle=':', linewidth=1.5,
                    label=f'Design Mode (n={design_mode})')
        ax3.axvline(2*design_mode, color='g', linestyle='-.', linewidth=1.5,
                    label='2nd Harmonic')

        ax3.set_xlim(0, 25)
        ax3.set_xlabel('Mode Number n', fontsize=12)
        ax3.set_ylabel('Normalized Power (log)', fontsize=12)
        ax3.set_title('Spectral Analysis of Curvature Profiles', fontsize=14)
        ax3.legend(fontsize=10)
        ax3.grid(True, which='both', alpha=0.3)

    def _plot_curvature_profiles(self):
        """Visualisiert die Krümmungsverläufe."""
        ax4 = plt.subplot2grid((3, 3), (0, 2), rowspan=3)

        for wave, color in zip(['hydrodynamic', 'electromagnetic'], ['b', 'r']):
            data = self.results[wave]
            ax4.plot(np.degrees(data['angle']),

    data['curvature']/np.max(data['curvature']),
                     color=color,
                     linewidth=1.5,
                     label=f"{data['type']}")

        ax4.set_xlabel('Azimuthal Angle [deg]', fontsize=12)
        ax4.set_ylabel('Normalized Curvature', fontsize=12)
        ax4.set_title('Curvature Profiles', fontsize=14)
        ax4.legend(fontsize=10)
        ax4.grid(True, alpha=0.3)
        ax4.set_xlim(0, 360)
```

```
252         ax4.set_xticks(np.arange(0, 361, 45))
253
254     def generate_report(self):
255         """Erstellt einen detaillierten Analysebericht mit
    physikalischer Interpretation."""
256         hd_mode =
    self.results['hydrodynamic']['dominant_mode']
257         em_mode =
    self.results['electromagnetic']['dominant_mode']
258         design_n = self.params['design_mode']['value']
259
260         print("\n" + "="*70)
261         print("                          WAVE ANALYSIS
    REPORT".center(70))
262         print("="*70)
263         print(f"{'Hydrodynamic Wave - Dominant Mode:':<40} n
    = {hd_mode:.1f}")
264         print(f"{'Electromagnetic Wave - Dominant
    Mode:':<40} n = {em_mode:.1f}")
265         print("="*70)
266
267         # Wissenschaftliche Interpretation basierend auf den
    Ergebnissen
268         if abs(hd_mode - 2*design_n) < 0.5 and abs(em_mode -
    design_n) < 0.5:
269             print("\nRESULT INTERPRETATION:")
270             print(f"- Hydrodynamic wave exhibits dominant
    2nd harmonic: n = {2*design_n}")
271             print("  This arises due to the nonlinear
    dependence of curvature on radius:")
272             print("  κ □ |r² + 2(r')² - r r''| / (r² +
    (r')²)^3/2 → enhances 2n components")
273             print(f"- Electromagnetic wave preserves
    fundamental mode: n = {design_n}")
274             print("  Smoothing from intensity thresholding
    and contour detection")
275             print("  suppresses higher harmonics, preserving
    design symmetry.")
276
277             print("\nSCIENTIFIC IMPLICATIONS:")
278             print("- Curvature acts as a nonlinear filter,
    amplifying even harmonics")
279             print("- Demonstrates fundamental difference
    between geometric modulation")
```

```
280             print("  and intensity-based wavefronts in mode
    detection")
281             print("- Validates theoretical models of
    modulated circular waves")
282             print("- Suggests curvature analysis is
    sensitive to nonlinear wave coupling")
283 
284             print("\nCONCLUSION:")
285             print("Both results are consistent with wave
    theory. The apparent 'discrepancy'")
286             print("reflects complementary physical
    behaviors, not an error.")
287 
288         elif abs(hd_mode - em_mode) > 1.0:
289             print("\nANALYSIS WARNING:")
290             print("Significant mode mismatch detected")
291             print("Recommended actions:")
292             print("- Verify numerical resolution (current:
    {})".format(self.params['resolution']['value']))
293             print("- Adjust contour threshold (current:
    {:.2f})".format(self.params['threshold']['value']))
294             print("- Check for aliasing or undersampling in
    curvature calculation")
295         else:
296             print("\nANALYSIS NOTE:")
297             print("Modes are consistent within expected
    tolerance.")
298             print("No further action required.")
299 
300         print("\nAnalysis completed successfully!")
301 
302 if __name__ == "__main__":
303     print("Initializing Wave Optics Analyzer...")
304     analyzer = WaveOpticsAnalyzer()
305 
306     try:
307         print("\nGenerating hydrodynamic wave profile...")
308         analyzer.generate_hydrodynamic_wave()
309 
310         print("Simulating electromagnetic wave...")
311         analyzer.generate_em_wave()
312 
313         print("\nPerforming spectral analysis...")
314         analyzer.perform_spectral_analysis()
```

```
315
316         print("\nGenerating visualizations...")
317         analyzer.visualize_results()
318
319         print("\nGenerating scientific report...")
320         analyzer.generate_report()
321
322         print("\nAnalysis completed successfully!")
323     except Exception as e:
324         print(f"\nERROR: {str(e)}")
325         print("\nTROUBLESHOOTING GUIDE:")
326         print("1. Adjust intensity threshold (current:
    {:.2f})".format(
327             analyzer.params['threshold']['value']))
328         print("2. Increase resolution (current: {})".format(
329             analyzer.params['resolution']['value']))
330         print("3. Check wave parameters (modulation, beam
    width)")
```

Listing A.2: Visualisierung Nichtlineare Krümmungskopplung in modulierten Kreiswellen

A.3 Optimale Pfadplanung, (Abschn. 8.0.3)

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from scipy.optimize import minimize
4  from scipy.integrate import cumulative_trapezoid, trapezoid
5
6  class CurvatureOptimalPlanner:
7      def __init__(self, kappa_max=0.4, path_length=18.0,
       n_modes=3):
8          """
9          Optimaler Pfadplaner mit Krümmungsbeschränkungen
10
11         Args:
12             kappa_max: Maximale Krümmung [1/m] (1/minimaler
       Wendekreis)
13             path_length: Gesamtpfadlänge [m]
14             n_modes: Anzahl der Fourier-Moden für
       Krümmungsmodulation
15         """
16         self.kappa_max = kappa_max
```

```
        self.path_length = path_length
        self.n_modes = n_modes

    def plan_path(self, start_pose=(0, 0, 0), goal_pose=(5, 5, 0), n_points=500):
        """
        Berechnet optimalen Pfad unter Krümmungsbeschränkungen

        Args:
            start_pose: (x, y, theta) Startposition [m] und Orientierung [rad]
            goal_pose: (x, y, theta) Zielposition [m] und Orientierung [rad]
            n_points: Anzahl der Diskretisierungspunkte

        Returns:
            tuple: (x_coords, y_coords, headings, curvatures)
        """
        s = np.linspace(0, self.path_length, n_points)

        # Parameterinitialisierung [kappa0, a1, b1, a2, b2, ...]
        initial_params = np.zeros(2 * self.n_modes + 1)
        param_bounds = [(-self.kappa_max, self.kappa_max)] + [(-1, 1)] * 2 * self.n_modes

        # Optimierung
        result = minimize(self._compute_cost, initial_params,
                          args=(s, start_pose, goal_pose),
                          bounds=param_bounds,
                          method='SLSQP')

        # Pfadgenerierung
        curvatures = self._compute_curvature_profile(result.x, s)
        x, y, theta = self._integrate_curvature(curvatures, s, start_pose)

        return x, y, theta, curvatures

    def _compute_curvature_profile(self, params, s):
        """Berechnet das modulierte Krümmungsprofil"""
        base_curvature = params[0]
```

```
        modulation = sum(
            a * np.cos(2 * np.pi * (i+1) * s / self.path_length) +
            b * np.sin(2 * np.pi * (i+1) * s / self.path_length)
            for i, (a, b) in enumerate(zip(params[1::2], params[2::2]))
        )
        return np.clip(base_curvature + modulation, -self.kappa_max, self.kappa_max)

    def _compute_cost(self, params, s, start_pose, goal_pose):
        """Berechnet die kombinierte Kostenfunktion"""
        curvatures = self._compute_curvature_profile(params, s)
        base_curvature = params[0]

        # Geometrische Wirkung (Defekt-Integral)
        defect_integral = trapezoid((curvatures - base_curvature)**2, s)

        # Pfadintegration
        x, y, theta = self._integrate_curvature(curvatures, s, start_pose)

        # Zielkosten (gewichtete quadratische Abweichungen)
        position_error = (x[-1] - goal_pose[0])**2 + (y[-1] - goal_pose[1])**2
        orientation_error = (theta[-1] - goal_pose[2])**2
        goal_cost = 10 * position_error + 7.0 * orientation_error

        return defect_integral + goal_cost

    def _integrate_curvature(self, curvatures, s, start_pose):
        """Integriert Krümmung zu kartesischen Koordinaten"""
        headings = start_pose[2] + cumulative_trapezoid(curvatures, s, initial=0)
        x_coords = start_pose[0] + cumulative_trapezoid(np.cos(headings), s, initial=0)
        y_coords = start_pose[1] + cumulative_trapezoid(np.sin(headings), s, initial=0)
```

```
        return x_coords, y_coords, headings

def visualize_path(x, y, curvatures, kappa_max, start_pose, goal_pose):
    """Erstellt wissenschaftliche Visualisierung der Ergebnisse"""
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))

    # Trajektorienplot
    ax1.plot(x, y, 'b-', linewidth=2, label='Optimaler Pfad')
    ax1.scatter([start_pose[0], goal_pose[0]],
                [start_pose[1], goal_pose[1]],
                c=['g', 'r'], s=100, label=['Start', 'Ziel'])
    ax1.quiver(*start_pose[:2], np.cos(start_pose[2]), np.sin(start_pose[2]),
               scale=10, color='g', width=0.005)
    ax1.quiver(x[-1], y[-1],
           np.cos(goal_pose[2]), np.sin(goal_pose[2]),
           color='m', scale=15, label='Soll-Orientierung')
    ax1.quiver(*goal_pose[:2], np.cos(goal_pose[2]), np.sin(goal_pose[2]),
               scale=10, color='r', width=0.005)
    ax1.set_title('Optimierte Trajektorie')
    ax1.set_xlabel('x [m]')
    ax1.set_ylabel('y [m]')
    ax1.axis('equal')
    ax1.grid(True)
    ax1.legend()

    # Krümmungsplot
    ax2.plot(np.linspace(0, len(x), len(curvatures)), curvatures, 'r-', label='Krümmung')
    ax2.axhline(kappa_max, color='k', linestyle='--', label='Maximale Krümmung')
    ax2.axhline(-kappa_max, color='k', linestyle='--')
    # Krümmungsplot mit Warnzone
    ax2.axhspan(0.475, 0.5, color='yellow', alpha=0.2, label='Kritische Krümmung')
    ax2.legend()
    ax2.set_title('Krümmungsverlauf')
    ax2.set_xlabel('Bogenlänge [m]')
    ax2.set_ylabel('κ [1/m]')
    ax2.grid(True)
    ax2.legend()
```

```
120
121     plt.tight_layout()
122     plt.savefig('sinus_kreis_pfadplanung_optimal.png',
    dpi=300)
123     plt.show()
124     plt.close()
125
126 if __name__ == "__main__":
127     # Systemparameter: Fahrzeug mit 2m minimalem Wendekreis
128     planner = CurvatureOptimalPlanner(
129         kappa_max=0.5,    # 1/2m = 0.5 m^-1
130         path_length=15.0,  # Gesamtpfadlänge
131         n_modes=5         # Fourier-Moden
132     )
133
134     # Randbedingungen
135     start = (0, 0, np.pi/4)   # (x, y, heading)
136     goal = (8, 6, -np.pi/2)
137
138     # Pfadberechnung
139     x, y, theta, kappa = planner.plan_path(start_pose=start,
    goal_pose=goal)
140
141     # Ergebnisanalyse
142     print("\n=== Pfadoptimierungsergebnisse ===")
143     print(f"Maximale Krümmung: {np.max(np.abs(kappa)):.3f}/m
    (Limit: {planner.kappa_max}/m)")
144     print(f"Defekt-Integral S:
    {trapezoid((kappa-np.mean(kappa))**2, np.linspace(0,
    planner.path_length, len(kappa))):.3f}")
145     print(f"Endpositionsfehler:
    {np.linalg.norm([x[-1]-goal[0], y[-1]-goal[1]]):.4f} m")
146     print(f"Orientierungsfehler:
    {np.abs(theta[-1]-goal[2]):.4f} rad")
147
148     # Visualisierung
149     visualize_path(x, y, kappa, planner.kappa_max, start,
    goal)
```

Listing A.3: Visualisierung

A.4 Doppelpendel mit geometrischer Resonanz, (Abschn. 9.2.1)

```python
# doppelpendel_geometrische_resonanz.py
# Einheitliche Analyse der geometrischen Dynamik des Doppelpendels
# - Robustheit über alle ICs + fokussierte Analyse regulärer Trajektorien
import numpy as np
from scipy.integrate import odeint
from scipy.signal import correlate, find_peaks
import matplotlib.pyplot as plt

# ------------------------------
# Physikalische Parameter
# ------------------------------
m1, m2 = 1.0, 1.0
l1, l2 = 1.0, 1.0
g = 9.81
t_max = 20.0
dt = 0.01
t = np.arange(0, t_max, dt)

# ------------------------------
# Doppelpendel-Gleichung
# ------------------------------
def double_pendulum(state, t, m1, m2, l1, l2, g):
    theta1, omega1, theta2, omega2 = state
    dtheta = theta1 - theta2
    c, s = np.cos(dtheta), np.sin(dtheta)

    denom = 2 * m1 + m2 - m2 * c**2
    if abs(denom) < 1e-10:
        denom = np.sign(denom) * 1e-10

    dtheta1_dt = omega1
    dtheta2_dt = omega2

    domega1_dt = (
        -g * (2*m1 + m2) * np.sin(theta1)
        - m2 * g * np.sin(theta1 - 2*theta2)
        - 2 * s * m2 * (omega2**2 * l2 + omega1**2 * l1 * c)
    ) / (l1 * denom)
```

```
39
40      domega2_dt = (
41          2 * s * (
42              omega1**2 * l1 * (m1 + m2)
43              + g * (m1 + m2) * np.cos(theta1)
44              + omega2**2 * l2 * m2 * c
45          )
46      ) / (l2 * denom)
47
48      return [dtheta1_dt, domega1_dt, dtheta2_dt, domega2_dt]
49
50  # ------------------------------
51  # Lyapunov-Exponent
52  # ------------------------------
53  def lyapunov_exponent(state0, t, perturb=1e-8):
54      sol = odeint(double_pendulum, state0, t, args=(m1, m2,
    l1, l2, g))
55      state0_pert = state0 + np.array([perturb, 0, perturb, 0])
56      sol_pert = odeint(double_pendulum, state0_pert, t,
    args=(m1, m2, l1, l2, g))
57      diff = np.linalg.norm(sol - sol_pert, axis=1)
58      diff = np.maximum(diff, 1e-16)
59      lyap = np.polyfit(t, np.log(diff), 1)[0]
60      return lyap, sol
61
62  # ------------------------------
63  # Geometrische Krümmung κ(t)
64  # ------------------------------
65  def compute_curvature(theta1, theta2, t):
66      f = np.sin(theta1) + np.sin(theta2)
67      df_dt = np.gradient(f, t)
68      d2f_dt2 = np.gradient(df_dt, t)
69      kappa = np.abs(d2f_dt2) / (1 + df_dt**2 + 1e-8)**1.5
70      return kappa
71
72  # ------------------------------
73  # Zeitliche Alignierung (zentriert)
74  # ------------------------------
75  def align_curves(curves, ref_idx=0):
76      ref = curves[ref_idx]
77      aligned = []
78      for curve in curves:
79          corr = correlate(curve - np.mean(curve), ref -
    np.mean(ref), mode='full')
```

```
        lag = np.argmax(corr) - len(ref) + 1  # zentrierte Lag-Berechnung
        if lag > 0:
            aligned_curve = np.concatenate([np.full(lag, curve[0]), curve[:-lag]])
        elif lag < 0:
            aligned_curve = np.concatenate([curve[-lag:], np.full(-lag, curve[-1])])
        else:
            aligned_curve = curve.copy()
        aligned.append(aligned_curve)
    return np.array(aligned)

# ------------------------------
# Anfangsbedingungen: alle
# ------------------------------
initial_conditions_all = [
    [np.pi/2, 0, np.pi/3, 0],           # IC 0
    [np.pi/2 + 0.1, 0, np.pi/3, 0],     # IC 1
    [np.pi/2, 0, np.pi/3 + 0.1, 0],     # IC 2
    [np.pi/2 - 0.1, 0.05, np.pi/3, 0],  # IC 3
    [1.0, 0, 1.0, 0],                   # IC 4
    [2.0, 0, 1.5, 0],                   # IC 5: chaotisch
    [np.pi - 0.1, 0, np.pi - 0.1, 0],   # IC 6: fast invertiert
]

# ------------------------------
# 1. ROBUSTHEITSANALYSE: alle ICs
# ------------------------------
print("🔍 Robustheitsanalyse: alle 7 Anfangsbedingungen")
results_all = []
for i, ic in enumerate(initial_conditions_all):
    lyap, sol = lyapunov_exponent(np.array(ic), t)
    theta1, theta2 = sol[:, 0], sol[:, 2]
    kappa = compute_curvature(theta1, theta2, t)
    S = np.sum((1.0 - kappa)**2 * dt)
    results_all.append({'idx': i, 'lyap': lyap, 'kappa': kappa, 'S': S})

# Klassifikation
threshold = 0.3
regular_indices = [i for i, r in enumerate(results_all) if r['lyap'] < threshold]
```

```
chaotic_indices = [i for i, r in enumerate(results_all) if r['lyap'] >= threshold]
print(f"✅ Regulär: {regular_indices}, ❌ Chaotisch: {chaotic_indices}")

# Alignierung aller
kappas_all = [r['kappa'] for r in results_all]
aligned_all = align_curves(kappas_all)
kappa_avg_all = np.mean(aligned_all, axis=0)

# ------------------------------
# 2. FOKUSSIERT: nur reguläre ICs
# ------------------------------
print("\n🔍 Fokussierte Analyse: nur reguläre Trajektorien")
initial_conditions_reg = [initial_conditions_all[i] for i in regular_indices]
results_reg = []
for i, ic in enumerate(initial_conditions_reg):
    lyap, sol = lyapunov_exponent(np.array(ic), t)
    theta1, theta2 = sol[:, 0], sol[:, 2]
    kappa = compute_curvature(theta1, theta2, t)
    results_reg.append({'kappa': kappa})

kappas_reg = [r['kappa'] for r in results_reg]
aligned_reg = align_curves(kappas_reg)
kappa_avg_reg = np.mean(aligned_reg, axis=0)

# Universelle Resonanzen (aus regulärer Analyse)
peaks_reg, _ = find_peaks(kappa_avg_reg, height=np.percentile(kappa_avg_reg, 95), distance=200)
t_peaks = t[peaks_reg]

# Schwebungshypothese: Korrelation
f1, f2 = 2.8, 2.1
beat_envelope = np.abs(np.cos(np.pi * (f1 - f2) * t))  # |cosπ(·0.7·t)|
correlation_r = np.corrcoef(kappa_avg_reg, beat_envelope)[0, 1]
print(f"📈 Korrelation mit Schwebungshüllkurve: r = {correlation_r:.3f}")

# ------------------------------
# PLOTS FÜR LATEX
# ------------------------------
```

```
155
156 # Plot A: S vs. Lyapunov (alle ICs)
157 plt.figure(figsize=(5, 4))
158 lyaps = [r['lyap'] for r in results_all]
159 S_vals = [r['S'] for r in results_all]
160 colors = ['green' if i in regular_indices else 'red' for i
        in range(len(lyaps))]
161 plt.scatter(lyaps, S_vals, c=colors, s=60, edgecolor='k')
162 for i, (l, s) in enumerate(zip(lyaps, S_vals)):
163     plt.text(l, s, f' {i}', fontsize=8)
164 plt.xlabel(r'Lyapunov-Exponent $\lambda$')
165 plt.ylabel(r'Defekt-Wirkung $S$')
166 plt.yscale('log')
167 plt.grid(True, alpha=0.3)
168 plt.tight_layout()
169 plt.savefig('doppelpendel_S_vs_lambda.png',
        bbox_inches='tight')
170 plt.close()
171
172 # Plot B: Mittlere Krümmung (regulär) + Resonanzen
173 plt.figure(figsize=(6, 3))
174 plt.plot(t, kappa_avg_reg, 'k-', linewidth=1.2,
        label=r'$\bar{\kappa}(t)$')
175 plt.plot(t_peaks, kappa_avg_reg[peaks_reg], 'ro',
        markersize=4, label='Resonanzen')
176 plt.xlabel(r'Zeit $t$ [s]')
177 plt.ylabel(r'Krümmung $\kappa(t)$')
178 plt.grid(True, alpha=0.3)
179 plt.legend(fontsize=9)
180 plt.tight_layout()
181 plt.savefig('doppelpendel_universal_peaks.png',
        bbox_inches='tight')
182 plt.close()
183
184 # Plot C: Schwebungshypothese
185 t_fine = np.linspace(0, t_max, len(t) * 5)
186 env_fine = np.abs(np.cos(np.pi * 0.7 * t_fine))
187 plt.figure(figsize=(6, 3))
188 plt.plot(t, kappa_avg_reg, 'k-', linewidth=1.2,
        label=r'$\bar{\kappa}(t)$')
189 plt.plot(t_fine, env_fine, 'C1--', alpha=0.8,
        label=r'Hüllkurve $|\cos(\pi \cdot 0.7 \cdot t)|$')
190 for tp in t_peaks:
```

```
    plt.axvline(tp, color='gray', linestyle=':',
    linewidth=0.8)
plt.xlabel(r'Zeit $t$ [s]')
plt.ylabel('Amplitude')
plt.legend(fontsize=9)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig('doppelpendel_beat_hypothesis.png',
    bbox_inches='tight')
plt.close()

# Plot D: Alle alignierten κ(t) (regulär)
plt.figure(figsize=(6, 3))
for k in aligned_reg:
    plt.plot(t, k, alpha=0.6, linewidth=0.8)
plt.plot(t, kappa_avg_reg, 'k-', linewidth=1.5,
    label='Mittelwert')
plt.xlabel(r'Zeit $t$ [s]')
plt.ylabel(r'$\kappa(t)$')
plt.grid(True, alpha=0.3)
plt.legend(fontsize=9)
plt.tight_layout()
plt.savefig('doppelpendel_all_aligned_reg.png',
    bbox_inches='tight')
plt.close()

print("\n Fertig! Plots als png für LaTeX gespeichert.")
```

Listing A.4: Visualisierung Doppelpendel mit geometrischer Resonanz

A.5 Kreiswelle mit Modenverstärkung (Animation), (Abschn. 12)

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

# Parameter
n = 8
eps_max = 0.5
frames = 50
```

```
N = 400
theta = np.linspace(0, 2 * np.pi, N, endpoint=False)
epsilons = np.linspace(0, eps_max, frames)

def curvature(r, th):
    dr = np.gradient(r, th)
    d2r = np.gradient(dr, th)
    num = np.abs(r**2 + 2*dr**2 - r*d2r)
    den = (r**2 + dr**2)**1.5
    return np.divide(num, den, out=np.zeros_like(num),
    where=den > 1e-10)

# Figur erstellen - bewusst etwas höher für Text
fig = plt.figure(figsize=(11, 6))  # Höhe leicht erhöht für
    besseren Abstand

# Titel (14 pt)
fig.suptitle('Nichtlineare Modenverstärkung', fontsize=14,
    fontweight='bold', y=0.96)

# Untertitel (12 pt) - unten, aber nicht zu tief
fig.text(0.5, 0.03, 'Visualisierung: Klaus H. Dieckmann,
    2025',
         ha='center', fontsize=12, style='italic')

# Dynamischer Erklärtext (oberhalb des Urheberhinweises)
explanation_text = fig.text(0.5, 0.08, '', ha='center',
    fontsize=12, fontweight='bold', color='darkblue')

# Zwei Subplots - explizit positioniert mit subplots_adjust
axs = fig.subplots(1, 2)

# Erster Plot
line1, = axs[0].plot([], [], 'b-', linewidth=1.8)
axs[0].set_aspect('equal')
axs[0].set_xlim(-1.8, 1.8)
axs[0].set_ylim(-1.8, 1.8)
axs[0].set_title('Wellenfront')
axs[0].grid(True, alpha=0.3)
axs[0].set_xticks([])
axs[0].set_yticks([])

# Zweiter Plot
line2, = axs[1].plot([], [], 'g-', linewidth=1.8)
```

```
49 axs[1].set_xlim(0, 2 * np.pi)
50 axs[1].set_ylim(0, 2.5)
51 axs[1].set_title('Krümmung κθ()')
52 axs[1].set_xlabel('θ [rad]')
53 axs[1].grid(True, alpha=0.3)
54
55 # WICHTIG: Explizite Platzierung der Plots - lässt Platz
       unten für Text!
56 plt.subplots_adjust(left=0.06, right=0.94, top=0.85,
       bottom=0.18, wspace=0.25)
57
58 # Phasenbasierte Erklärtexte
59 def get_explanation_text(i, total_frames):
60     phase = i / total_frames
61     if phase < 0.2:
62         return "Initiale Kreisform: ungestörte Welle mit
       konstanter Krümmung."
63     elif phase < 0.4:
64         return "Leichte Modulation: sinusförmige Verformung
       beginnt."
65     elif phase < 0.6:
66         return "Nichtlineare Verstärkung: Krümmung an
       Spitzen nimmt stark zu."
67     elif phase < 0.8:
68         return "Maximale Deformation: scharfe
       Krümmungsmaxima bei n-facher Symmetrie."
69     else:
70         return "Rückkehr zur Symmetrie: Modulation klingt
       ab, Krümmung glättet sich."
71
72 def animate(i):
73     eps = epsilons[i]
74     r = 1.0 + eps * np.sin(n * theta)
75     x = r * np.cos(theta)
76     y = r * np.sin(theta)
77     kappa = curvature(r, theta)
78
79     line1.set_data(x, y)
80     line2.set_data(theta, kappa)
81     explanation_text.set_text(get_explanation_text(i,
       frames))
82
83     return line1, line2, explanation_text
84
```

```
# Animation - blit=False, da wir Text außerhalb der Achsen aktualisieren
anim = FuncAnimation(fig, animate, frames=frames, interval=150, blit=False)

# GIF speichern - OHNE bbox_inches, um Text nicht abzuschneiden
anim.save('nichtlineare_modenverstaerkung_animation.gif', writer='pillow', fps=5, dpi=100)

plt.show()
```

Listing A.5: Visualisierung Kreiswelle mit Modenverstärkung (Animation)

A.6 Windverformung einer Kreiswelle (Animation), (Abschn. 10)

```
# windverformung_kreiswelle_animation.py
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from matplotlib.collections import LineCollection

# ----------------------------
# Parameter
# ----------------------------
n = 8
A_max = 0.6
omega = np.sqrt(1 / A_max)  # Damit A_max * omega^2 = 1 → Resonanz
frames = 60
N = 500
theta = np.linspace(0, 2 * np.pi, N, endpoint=False)
As = np.linspace(0, A_max, frames)

# ----------------------------
# Hilfsfunktionen
# ----------------------------
def curvature(r, th):
    dr = np.gradient(r, th)
    d2r = np.gradient(dr, th)
```

```
    num = r**2 + 2*dr**2 - r*d2r
    den = (r**2 + dr**2)**1.5
    kappa = np.divide(num, den, out=np.ones_like(num),
   where=den > 1e-10)
    return kappa

def curvature_defect(kappa, ds):
    return (1 - kappa) * ds

# ----------------------------
# Setup Plot
# ----------------------------
fig, ax = plt.subplots(1, 1, figsize=(8, 8))
fig.suptitle('Geometrische Resonanz: Windverformung einer
   Kreiswelle', fontsize=14, fontweight='bold')
fig.text(0.5, 0.06, 'Visualisierung: Klaus H. Dieckmann,
   2025', ha='center', fontsize=11, style='italic')
info_text = fig.text(0.5, 0.9, '', ha='center', fontsize=12,
   color='darkred')

ax.set_aspect('equal')
ax.set_xlim(-2.0, 2.0)
ax.set_ylim(-2.0, 2.0)
ax.axis('off')

# Windpfeile
wind_arrow1 = ax.annotate('', xy=(1.6, 0.3), xytext=(0.8,
   0.3),
                          arrowprops=dict(arrowstyle='->',
   color='gray', lw=1.5, ls='--'))
wind_arrow2 = ax.annotate('', xy=(1.6, -0.3), xytext=(0.8,
   -0.3),
                          arrowprops=dict(arrowstyle='->',
   color='gray', lw=1.5, ls='--'))
wind_arrow1.set_visible(False)
wind_arrow2.set_visible(False)

# Globale Referenz für die Linie (wird in animate
   aktualisiert)
current_lc = None

# ----------------------------
# Animation
# ----------------------------
```

```
def animate(i):
    global current_lc

    A = As[i]
    r = 1.0 + A * np.sin(n * theta)
    x = r * np.cos(theta)
    y = r * np.sin(theta)

    # Krümmung & Defekt
    kappa = curvature(r, theta)
    dx = np.gradient(x, theta)
    dy = np.gradient(y, theta)
    ds = np.sqrt(dx**2 + dy**2)
    dDelta = curvature_defect(kappa, ds)

    # Farben
    norm = plt.Normalize(vmin=-0.8, vmax=0.8)
    colors = plt.cm.coolwarm(norm(dDelta))

    # Neue LineCollection
    points = np.array([x, y]).T.reshape(-1, 1, 2)
    segments = np.concatenate([points[:-1], points[1:]],
    axis=1)
    new_lc = LineCollection(segments, colors=colors,
    linewidths=2.5)

    # Alte Linie entfernen (wenn vorhanden)
    if current_lc is not None:
        current_lc.remove()

    # Neue Linie hinzufügen
    ax.add_collection(new_lc)
    current_lc = new_lc

    # Windpfeile sichtbar ab Frame 10
    wind_arrow1.set_visible(i >= 10)
    wind_arrow2.set_visible(i >= 10)

    # Info-Text
    A_omega2 = A * omega**2
    if i < 5:
        txt = "Unverformte Kreiswelle: κ = 1 überall."
    elif i < 15:
```

```
        txt = f"Wind setzt ein → Verformung beginnt. ωA·² = {A_omega2:.2f}"
    else:
        resonance = "Resonanz!" if abs(A_omega2 - 1.0) < 0.05 else ""
        txt = f"Sinusförmige Verformung (n={n}). ωA·² = {A_omega2:.2f} {resonance}"
    info_text.set_text(txt)

    return new_lc, wind_arrow1, wind_arrow2, info_text

# ----------------------------
# Start & Speichern
# ----------------------------
anim = FuncAnimation(fig, animate, frames=frames, interval=120, blit=False)

# Wichtig: Kein tight_layout nach FuncAnimation – kann zu Leaks führen
plt.subplots_adjust(left=0.05, right=0.95, top=0.88, bottom=0.08)

anim.save('windverformung_kreiswelle_animation.gif', writer='pillow', fps=5, dpi=110)

plt.show()
```

Listing A.6: Visualisierung Windverformung einer Kreiswelle (Animation)

A.7 Roboter-Trajektorie (Animation), (Abschn. 8)

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

# ----------------------------
# Parameter
# ----------------------------
x_max = 4 * np.pi
N = 500
frames = 70
```

```
switch_frame = 35  # Umschaltung in der Mitte

x = np.linspace(0, x_max, N)
dx = x[1] - x[0]

# Sinus-Trajektorie
y_sin = np.sin(x)
dy_sin = np.gradient(y_sin, dx)
d2y_sin = np.gradient(dy_sin, dx)
kappa_sin = np.abs(d2y_sin) / (1 + dy_sin**2)**1.5

# Defekt-minimierte Näherung (glattere Krümmung)
y_opt = 0.95 * np.sin(x * 0.88)
dy_opt = np.gradient(y_opt, dx)
d2y_opt = np.gradient(dy_opt, dx)
kappa_opt = np.abs(d2y_opt) / (1 + dy_opt**2)**1.5

# ----------------------------
# Setup Plot (2 Reihen)
# ----------------------------
fig, axs = plt.subplots(2, 1, figsize=(9, 6),
    gridspec_kw={'height_ratios': [3, 2]})
fig.suptitle('Geometrische Resonanz\nRobotertrajektorie:
    Sinus vs. defekt-minimiert', fontsize=14,
    fontweight='bold')
fig.text(0.5, 0.015, 'Visualisierung: Klaus H. Dieckmann,
    2025', ha='center', fontsize=11, style='italic')

# Dynamischer Erklärtext (wird in animate aktualisiert)
explanation_text = fig.text(0.5, 0.06, '', ha='center',
    fontsize=12, color='darkblue', fontweight='bold')

# Plot 1: Trajektorie
axs[0].plot(x, y_sin, 'lightgray', lw=1,
    label='Sinus-Trajektorie')
axs[0].plot(x, y_opt, 'lightgray', lw=1,
    label='Defekt-minimiert')
axs[0].set_xlim(0, x_max)
axs[0].set_ylim(-1.6, 1.6)
axs[0].set_ylabel('y')
axs[0].grid(True, alpha=0.3)
axs[0].legend(loc='upper right', fontsize=8)
robot_point, = axs[0].plot([], [], 'ro', markersize=7)

```

```
# Plot 2: Krümmung
axs[1].plot(x, kappa_sin, 'b--', lw=1.2,
    label=r'$\kappa_{\sin}(x)$')
axs[1].plot(x, kappa_opt, 'g--', lw=1.2,
    label=r'$\kappa_{\text{opt}}(x)$')
axs[1].set_xlim(0, x_max)
axs[1].set_ylim(0, np.max(kappa_sin) * 1.1)
axs[1].set_xlabel('x')
axs[1].set_ylabel(r'$\kappa$')
axs[1].grid(True, alpha=0.3)
axs[1].legend(loc='upper right', fontsize=8)
kappa_point, = axs[1].plot([], [], 'ro', markersize=5)

# Abstand sichern - Platz für Erklärtext & Urheberhinweis
plt.subplots_adjust(left=0.08, right=0.95, top=0.88,
    bottom=0.17, hspace=0.35)

# Globale Referenz für aktive (farbige) Trajektorienlinie
active_traj_line = None

# ----------------------------
# Phasenbasierte Erklärtexte
# ----------------------------
def get_explanation(i, switch=switch_frame):
    if i < 5:
        return "Start: Roboter folgt klassischer
    Sinus-Trajektorie."
    elif i < switch - 5:
        return "Problem: Krümmung zu gering bei Nullstellen
    → ineffiziente Lenkbewegung."
    elif i < switch + 5:
        return "→ Umschaltung auf defekt-minimierte Bahn
    (konstante Krümmung)."
    elif i < frames - 10:
        return "Optimierte Bahn: Krümmung nahezu konstant →
    maximale Befahrbarkeit."
    else:
        return "Vergleich abgeschlossen: Defekt-minimierte
    Trajektorie reduziert geometrischen Verschleiß."

# ----------------------------
# Animation
# ----------------------------
def animate(i):
```

```
    global active_traj_line

    idx = min(i * (N // frames), N - 1)
    use_opt = i >= switch_frame

    x_now = x[idx]
    y_now = y_opt[idx] if use_opt else y_sin[idx]
    kappa_now = kappa_opt[idx] if use_opt else kappa_sin[idx]

    # Roboterposition
    robot_point.set_data([x_now], [y_now])
    kappa_point.set_data([x_now], [kappa_now])

    # Alte aktive Linie entfernen
    if active_traj_line is not None:
        active_traj_line.remove()
        active_traj_line = None

    # Neue aktive Linie zeichnen
    y_traj = y_opt if use_opt else y_sin
    color = 'green' if use_opt else 'blue'
    active_traj_line = axs[0].plot(x[:idx+1],
    y_traj[:idx+1], color=color, lw=2.2)[0]

    # Dynamischen Text aktualisieren
    explanation_text.set_text(get_explanation(i))

    return robot_point, kappa_point, active_traj_line,
    explanation_text

# ----------------------------
# Speichern
# ----------------------------
anim = FuncAnimation(fig, animate, frames=frames,
    interval=140, blit=False)
anim.save('roboter_trajektorie_animation.gif',
    writer='pillow', fps=5, dpi=110)

plt.show()
```

Listing A.7: Visualisierung

A.8 Projektion Kreis: Sinus mit Krümmungsvergleich (Animation), (Abschn. 4)

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

# ----------------------------
# Parameter
# ----------------------------
x_max = 4 * np.pi
N_total = 600
frames = 100

theta_full = np.linspace(0, x_max, N_total)
x_full = theta_full
y_sin_full = np.sin(x_full)

dx = x_full[1] - x_full[0]
dy = np.gradient(y_sin_full, dx)
d2y = np.gradient(dy, dx)
kappa_sin = np.abs(d2y) / (1 + dy**2)**1.5

# Resonanzstellen: wo κ ≈ 1 (nahe Maxima)
resonance_mask = np.isclose(kappa_sin, 1.0, atol=0.03)
x_res = x_full[resonance_mask]
kappa_res = kappa_sin[resonance_mask]

# ----------------------------
# Figur & Layout
# ----------------------------
fig = plt.figure(figsize=(10, 6.2))
fig.suptitle('Geometrische Resonanz\nProjektion Kreis: Sinus mit Krümmungsvergleich', fontsize=14, fontweight='bold')
fig.text(0.18, 0.26, 'Visualisierung: Klaus H. Dieckmann, 2025', ha='center', fontsize=11, style='italic')

# Dynamischer Erklärtext (wird in animate aktualisiert)
explanation_text = fig.text(0.5, 0.03, '', ha='center', fontsize=12, color='darkblue', fontweight='bold')

# Plots manuell positionieren (mehr Kontrolle)
```

```
ax_circle = plt.axes([0.05, 0.34, 0.25, 0.52])   # Kreis links
ax_sin    = plt.axes([0.38, 0.34, 0.57, 0.52])   # Sinus rechts
ax_kappa  = plt.axes([0.38, 0.12, 0.57, 0.20])   # Krümmung unten

# --- Kreis ---
ax_circle.set_aspect('equal')
ax_circle.set_xlim(-1.3, 1.3)
ax_circle.set_ylim(-1.3, 1.3)
ax_circle.set_title('Einheitskreis')
ax_circle.grid(True, alpha=0.3)
ax_circle.set_xticks([-1, 0, 1])
ax_circle.set_yticks([-1, 0, 1])
circle_theta = np.linspace(0, 2*np.pi, 200)
ax_circle.plot(np.cos(circle_theta), np.sin(circle_theta), 'k-', lw=0.8)
point_circle, = ax_circle.plot([], [], 'ro', markersize=8)
proj_line, = ax_circle.plot([], [], 'r--', lw=1)

# --- Sinus ---
ax_sin.set_xlim(0, x_max)
ax_sin.set_ylim(-1.4, 1.4)
ax_sin.set_title('Sinusfunktion $y = \\sin x$')
ax_sin.grid(True, alpha=0.3)
ax_sin.set_ylabel('y')
sin_point, = ax_sin.plot([], [], 'ro', markersize=8)

# --- Krümmung ---
ax_kappa.set_xlim(0, x_max)
ax_kappa.set_ylim(0, np.max(kappa_sin)*1.1)
ax_kappa.set_xlabel('x')
ax_kappa.set_ylabel(r'$\kappa(x)$')
ax_kappa.grid(True, alpha=0.3)
kappa_point, = ax_kappa.plot([], [], 'ro', markersize=5)
resonance_scatter = ax_kappa.scatter([], [], c='red', s=20, zorder=5, label=r'$\kappa = 1$')
ax_kappa.legend(loc='upper right', fontsize=8)

# Globale Linien (für stabile Animation)
current_sin_line = None
current_kappa_line = None

```

```
# ----------------------------
# Dynamischer Erklärtext je Phase
# ----------------------------
def get_explanation(i, total_frames):
    phase = i / total_frames
    if phase < 0.1:
        return "Start: Punkt bewegt sich gleichförmig auf dem Einheitskreis κ( = 1)."
    elif phase < 0.3:
        return "Horizontale Projektion erzeugt die Sinusfunktion y = sin(x)."
    elif phase < 0.6:
        return "Die Sinuskurve hat nicht konstante Krümmung, im Gegensatz zum Kreis."
    elif phase < 0.85:
        return "Nur an den Maxima (x = π/2, π5/2, …) gilt κ = 1 → lokale Kreisähnlichkeit."
    else:
        return "Resonanzpunkte markiert: dort ist die geometrische Struktur identisch zum Kreis."

# ----------------------------
# Animation
# ----------------------------
def animate(i):
    global current_sin_line, current_kappa_line

    idx = min(i * (N_total // frames), N_total - 1)
    if idx < 0:
        idx = 0

    theta = theta_full[idx]
    x_now = x_full[idx]
    y_now = y_sin_full[idx]
    k_now = kappa_sin[idx]

    # Kreis
    point_circle.set_data([np.cos(theta)], [np.sin(theta)])
    proj_line.set_data([np.cos(theta), np.cos(theta)], [np.sin(theta), y_now])

    # Sinus-Linie
    if current_sin_line is not None:
        current_sin_line.remove()
```

```
    current_sin_line = ax_sin.plot(x_full[:idx+1], y_sin_full[:idx+1], 'b-', lw=2)[0]
    sin_point.set_data([x_now], [y_now])

    # Krümmung
    if current_kappa_line is not None:
        current_kappa_line.remove()
    current_kappa_line = ax_kappa.plot(x_full[:idx+1], kappa_sin[:idx+1], 'g-', lw=2)[0]
    kappa_point.set_data([x_now], [k_now])

    # Resonanzpunkte einblenden
    visible_res_x = x_res[x_res <= x_now]
    visible_res_k = kappa_res[x_res <= x_now]

    resonance_scatter.set_offsets(np.column_stack([visible_res_x, visible_res_k]))

    # Dynamischen Text aktualisieren
    explanation_text.set_text(get_explanation(i, frames))

    return (point_circle, proj_line, sin_point, kappa_point, resonance_scatter,
            current_sin_line, current_kappa_line, explanation_text)

# ----------------------------
# Speichern
# ----------------------------
anim = FuncAnimation(fig, animate, frames=frames, interval=100, blit=False)
anim.save('projektion_kreis_sinus_animation.gif', writer='pillow', fps=5, dpi=110)

plt.show()
```

Listing A.8: Visualisierung Projektion Kreis: Sinus mit Krümmungsvergleich (Animation)

A.9 Doppelpendel (Animation), (Abschn. 9)

```
# doppelpendel_chaos_geometrie.py
import numpy as np
```

```
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from scipy.integrate import solve_ivp
from scipy.signal import find_peaks

# ----------------------------
# Doppelpendel-Gleichungen
# ----------------------------
def double_pendulum(t, y, L1=1.0, L2=1.0, m1=1.0, m2=1.0, g=9.81):
    theta1, z1, theta2, z2 = y
    c, s = np.cos(theta1 - theta2), np.sin(theta1 - theta2)
    denom = m1 + m2 * s**2
    dtheta1 = z1
    dtheta2 = z2
    dz1 = (m2 * g * np.sin(theta2) * c - m2 * s * (L1 * z1**2 * c + L2 * z2**2) -
           (m1 + m2) * g * np.sin(theta1)) / (L1 * denom)
    dz2 = ((m1 + m2) * (L1 * z1**2 * s - g * np.sin(theta2) + g * np.sin(theta1) * c) +
           m2 * L2 * z2**2 * s * c) / (L2 * denom)
    return [dtheta1, dz1, dtheta2, dz2]

# ----------------------------
# Simulation: speichere θ1, θ2, f, κ
# ----------------------------
t_max = 18
dt = 0.02
t_eval = np.arange(0, t_max, dt)
n_runs = 3
y0_base = [np.pi / 2, 0, np.pi / 2, 0]

all_data = []  # Liste von dicts

for i in range(n_runs):
    y0 = y0_base + np.random.normal(0, 1e-4, 4)
    sol = solve_ivp(double_pendulum, [0, t_max], y0, t_eval=t_eval, rtol=1e-8, atol=1e-8)
    theta1, theta2 = sol.y[0], sol.y[2]
    f = np.sin(theta1) + np.sin(theta2)
    # Krümmung von (t, f(t))
    df = np.gradient(f, dt)
    d2f = np.gradient(df, dt)
    kappa = np.abs(d2f) / (1 + df**2)**1.5
```

```
    all_data.append({
        't': t_eval,
        'theta1': theta1,
        'theta2': theta2,
        'f': f,
        'kappa': kappa
    })

# Mittelung
kappas = np.array([d['kappa'] for d in all_data])
kappa_mean = np.mean(kappas, axis=0)
window = 61
kappa_smooth = np.convolve(kappa_mean,
    np.ones(window)/window, mode='same')
peaks, _ = find_peaks(kappa_smooth,
    height=np.percentile(kappa_smooth, 92), distance=120)

# ----------------------------
# Plot Setup
# ----------------------------
frames = 100
fig = plt.figure(figsize=(10, 7))
fig.suptitle('Doppelpendel: Chaos vs. geometrische Ordnung',
    fontsize=14, fontweight='bold')
fig.text(0.2, 0.3, 'Visualisierung:\nKlaus H. Dieckmann,
    2025', ha='center', fontsize=10, style='italic')
explanation_text = fig.text(0.65, 0.06, '', ha='center',
    fontsize=11, fontweight='bold', color='darkblue')

ax_pend = plt.axes([0.05, 0.55, 0.25, 0.35])
ax_pend.set_xlim(-2.5, 2.5)
ax_pend.set_ylim(-2.5, 2.5)
ax_pend.set_aspect('equal')
ax_pend.axis('off')
ax_pend.set_title('Doppelpendel', fontsize=10)

ax_f = plt.axes([0.38, 0.55, 0.57, 0.35])
ax_f.set_xlim(0, t_max)
ax_f.set_ylim(-2.1, 2.1)
ax_f.set_ylabel('f(t)')
ax_f.grid(True, alpha=0.3)
ax_f.set_title(r'$f(t) = \sin\theta_1 + \sin\theta_2$',
    fontsize=10)
line_f, = ax_f.plot([], [], 'b-', lw=1.2)
```

```
81
82 ax_k = plt.axes([0.38, 0.17, 0.57, 0.30])
83 ax_k.set_xlim(0, t_max)
84 ax_k.set_ylim(0, np.max(kappa_smooth)*1.15)
85 ax_k.set_xlabel('t')
86 ax_k.set_ylabel(r'$\kappa(t)$')
87 ax_k.grid(True, alpha=0.3)
88 ax_k.set_title('Geglättete Krümmung → universelle
    Resonanzen', fontsize=10)
89 line_k_raw, = ax_k.plot([], [], 'lightgray', lw=0.7)
90 line_k_smooth, = ax_k.plot([], [], 'g-', lw=2)
91 peak_scatter = ax_k.scatter([], [], c='red', s=30, zorder=5,
    label='Resonanzpeaks')
92 ax_k.legend(fontsize=8)
93
94 # ----------------------------
95 # Pendel zeichnen
96 # ----------------------------
97 def draw_pendulum(ax, theta1, theta2, L1=1.0, L2=1.0):
98     ax.clear()
99     ax.set_xlim(-2.5, 2.5)
100    ax.set_ylim(-2.5, 2.5)
101    ax.set_aspect('equal')
102    ax.axis('off')
103    x0, y0 = 0, 0
104    x1 = L1 * np.sin(theta1)
105    y1 = -L1 * np.cos(theta1)
106    x2 = x1 + L2 * np.sin(theta2)
107    y2 = y1 - L2 * np.cos(theta2)
108    ax.plot([x0, x1], [y0, y1], 'o-', color='C0', lw=2.5,
    markersize=8)
109    ax.plot([x1, x2], [y1, y2], 'o-', color='C1', lw=2.5,
    markersize=8)
110
111 # ----------------------------
112 # Animation
113 # ----------------------------
114 def animate(i):
115     idx = min(i * (len(t_eval) // frames), len(t_eval) - 1)
116     if idx < 20:
117         idx = 20
118
119     # Daten des ersten Laufs für Pendel
120     theta1 = all_data[0]['theta1'][idx]
```

```
    theta2 = all_data[0]['theta2'][idx]
    draw_pendulum(ax_pend, theta1, theta2)

    # f(t)
    t_show = t_eval[:idx+1]
    f_show = all_data[0]['f'][:idx+1]
    line_f.set_data(t_show, f_show)

    # Krümmung (Rohdaten aller Läufe bis idx)
    k_raw_show = np.mean([d['kappa'][:idx+1] for d in
    all_data], axis=0)
    line_k_raw.set_data(t_show, k_raw_show)

    # Geglättete Krümmung (bis idx)
    k_smooth_show = kappa_smooth[:idx+1]
    line_k_smooth.set_data(t_show, k_smooth_show)

    # Resonanzpeaks bis aktueller Zeitpunkt
    peaks_now = peaks[peaks <= idx]
    if len(peaks_now) > 0:

    peak_scatter.set_offsets(np.column_stack([t_eval[peaks_now],
    kappa_smooth[peaks_now]]))
    else:
        peak_scatter.set_offsets(np.empty((0, 2)))

    # Erklärtext
    explanation_text.set_text(get_explanation(i, frames))

    return line_f, line_k_raw, line_k_smooth, peak_scatter

def get_explanation(i, total_frames):
    phase = i / total_frames
    if phase < 0.25:
        return "Chaotische Bewegung des Doppelpendels,
    deterministisch, aber unvorhersagbar."
    elif phase < 0.5:
        return "Die Funktion f(t) = θ₁sin + θ₂sin oszilliert
    komplex und aperiodisch."
    elif phase < 0.75:
        return "Krümmung κ(t) zeigt verborgene Struktur.
    Rauschen überlagert Resonanzen."
    else:
```

```
158         return "Mittelung über mehrere Läufe enthüllt
    universelle, periodische Resonanzpeaks!"
159
160 # ----------------------------
161 # Speichern
162 # ----------------------------
163 anim = FuncAnimation(fig, animate, frames=frames,
    interval=120, blit=False)
164 anim.save('doppelpendel_chaos_animation.gif',
    writer='pillow', fps=5, dpi=110)
165
166 plt.show()
```

Listing A.9: Visualisierung Doppelpendel (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: 1. Oktober 2025
TimeStamp: https://freetsa.org/index_de.php

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

Literatur

[1] L. Allen, M. W. Beijersbergen, R. J. C. Spreeuw, and J. P. Woerdman. Orbital angular momentum of light and the transformation of Laguerre-Gaussian laser modes. *Physical Review A*, 45(11):8185–8189, 1992.

[2] M. P. do Carmo. *Differential Geometry of Curves and Surfaces*. Dover, 2nd edition. 2016.

[3] L. D. Landau und E. M. Lifshitz. *Fluidmechanik*. Pergamon Press, 2nd editio. 1987.

[4] S. A. Maier, *Plasmonics: Fundamentals and Applications*. Springer, 2007.

[5] D. J. Schwarz, G. D. Starkman, C. J. Copi, and P. M. Vaudrevange. CMB anomalies: Evidence for physics beyond the standard model? *arXiv preprint* arXiv:1510.07929 [astro-ph.CO], 2015.