Beispiele

Beispiele für Fourier-Reihen

Die Fourier-Reihen-Entwicklung periodischer Funktionen lässt sich an konkreten Beispielen veranschaulichen. In der praktischen Anwendung werden diese Entwicklungen häufig numerisch mit der Fast Fourier Transform (FFT) berechnet, die eine effiziente Implementierung der diskreten Fourier-Transformation darstellt.

Beispiel: Rechteckwelle

Eine periodische Rechteckwelle mit Periode L und Amplitude A dient als klassisches Beispiel für die Fourier-Entwicklung. Die Rechteckwelle springt abrupt zwischen zwei Werten und besitzt daher Unstetigkeiten, die sich in einem charakteristischen Konvergenzverhalten der Fourier-Reihe widerspiegeln.

Die Fourier-Reihe der Rechteckwelle lautet:

$$f(x) = \frac{4A}{\pi} \sum_{n=1,3,5,\ldots}^{\infty} \frac{1}{n} \sin\left(\frac{2\pi n x}{L}\right)$$

Diese Darstellung zeigt, dass nur die ungeraden Harmonischen zum Signal beitragen. Die Koeffizienten fallen mit $1/n$ ab, was auf die Unstetigkeiten der Funktion zurückzuführen ist. Je höher die Ordnung der harmonischen Komponente, desto geringer ihr Beitrag zur Gesamtfunktion.

%## Beispiel: Sägezahnwelle % %Die Sägezahnwelle steigt linear an und fällt dann abrupt %auf ihren Ausgangswert zurück. Diese Funktion besitzt %eine Fourier-Darstellung, die alle harmonischen %Komponenten einschließt: %

% %Im Gegensatz zur Rechteckwelle treten hier sowohl gerade %als auch ungerade Harmonische auf. Die alternierenden %Vorzeichen bewirken eine teilweise Auslöschung %benachbarter Terme, was zur Form der Sägezahnwelle führt.

%## Beispiel: Dreieckswelle % %Die Dreieckswelle unterscheidet sich von der %Sägezahnwelle durch ihre symmetrische Form. Sie steigt %linear an, erreicht ein Maximum und fällt dann linear %wieder ab. Ihre Fourier-Reihe konvergiert schneller als %die der Rechteckwelle: %

%Die Koeffizienten fallen hier mit $1/n^2$ ab statt mit %$1/n$ wie bei der Rechteckwelle. Dieser schnellere %Abfall resultiert aus der Tatsache, dass die %Dreieckswelle stetig ist und nur ihre Ableitung Sprünge %aufweist. Die Stetigkeit einer Funktion führt generell %zu einem schnelleren Abfall der Fourier-Koeffizienten.

Python-Implementierung

Die numerische Berechnung und Visualisierung von Fourier-Reihen lässt sich mit Python durchführen. Das folgende Programm approximiert eine Rechteckwelle durch eine endliche Fourier-Reihe.

```
import numpy as np
import matplotlib.pyplot as plt

# Beispiel: Fourier-Reihe einer Rechteckwelle
def square_wave_fourier(x, L, A, N_terms):
    """
    Approximiert eine Rechteckwelle durch Fourier-Reihe.

    Parameters:
    x: Array der x-Werte
    L: Periode
    A: Amplitude
    N_terms: Anzahl der Terme in der Reihe
    """
    f = np.zeros_like(x)
    for n in range(1, N_terms+1, 2):  # nur ungerade n
        f += (4*A/(np.pi*n)) * np.sin(2*np.pi*n*x/L)
    return f

# Visualisierung
x = np.linspace(0, 2, 1000)
L = 1.0
A = 1.0
```

```
fig, axes = plt.subplots(2, 2, figsize=(7, 4))

for i, N in enumerate([1, 3, 10, 50]):
    ax = axes[i//2, i%2]
    f_approx = square_wave_fourier(x, L, A, N)
    ax.plot(x, f_approx, label=f'N = {N} Terme')
    ax.set_xlabel('x')
    ax.set_ylabel('f(x)')
    ax.set_title(f'Fourier-Approximation mit {N} Termen')
    ax.grid(True, alpha=0.3)
    ax.legend()
    ax.set_ylim([-1.5, 1.5])

plt.tight_layout()
plt.show()
```

Die Visualisierung zeigt die Konvergenz der Fourier-Reihe mit zunehmender Anzahl von Termen. Mit nur einem Term erhält man eine grobe Annäherung durch eine einfache Sinusfunktion. Mit drei Termen beginnt sich bereits die charakteristische Form der Rechteckwelle abzuzeichnen. Mit zehn Termen ist die Approximation deutlich besser, und mit fünfzig Termen ist die Rechteckwelle nahezu perfekt reproduziert. An den Unstetigkeiten tritt jedoch das Gibbs-Phänomen auf, ein charakteristisches Überschwingen, das auch bei beliebig vielen Termen nicht verschwindet.

Fast Fourier Transform (FFT)

Die Fast Fourier Transform ist ein effizienter Algorithmus zur Berechnung der diskreten Fourier-Transformation. Statt der naiven Implementierung mit $O(N^2)$ Rechenoperationen benötigt die FFT nur $O(N \log N)$ Operationen. Diese dramatische Reduktion der Rechenlast macht die FFT zu einem der wichtigsten Algorithmen der numerischen Mathematik und ermöglicht die Echtzeitverarbeitung von Signalen.

Das folgende Beispiel demonstriert die Anwendung der FFT auf ein synthetisches Signal, das aus zwei überlagerten Sinusschwingungen besteht.

```
import numpy as np
import matplotlib.pyplot as plt

# Signal erzeugen
t = np.linspace(0, 1, 1000, endpoint=False)
f = np.sin(2*np.pi*50*t) + 0.5*np.sin(2*np.pi*120*t)  # 50 Hz + 120 Hz
```

```
# FFT berechnen
F = np.fft.fft(f)
freqs = np.fft.fftfreq(len(t), t[1]-t[0])

# Plotten
fig, axes = plt.subplots(1, 2, figsize=(7, 4))

# Zeitsignal
axes[0].plot(t, f)
axes[0].set_xlabel('Zeit [s]')
axes[0].set_ylabel('Amplitude')
axes[0].set_title('Zeitsignal')
axes[0].grid(True, alpha=0.3)

# Frequenzspektrum
axes[1].plot(freqs[:len(freqs)//2], np.abs(F)[:len(F)//2])
axes[1].set_xlabel('Frequenz [Hz]')
axes[1].set_ylabel('Amplitude')
axes[1].set_title('Frequenzspektrum (FFT)')
axes[1].grid(True, alpha=0.3)
axes[1].set_xlim([0, 200])

plt.tight_layout()
plt.show()
```

Das Zeitsignal zeigt eine komplexe Überlagerung zweier Frequenzen, die im Zeitbereich schwer zu erkennen sind. Das Frequenzspektrum macht hingegen die beiden Komponenten bei 50 Hz und 120 Hz unmittelbar sichtbar. Die FFT zerlegt das Signal in seine spektralen Bestandteile und liefert damit Aufschluss über die im Signal enthaltenen Frequenzen und ihre jeweiligen Amplituden.

Anwendung: Filterung im Frequenzbereich

Die Fourier-Transformation ermöglicht die Filterung von Signalen durch Manipulation im Frequenzbereich. Ein idealer Tiefpassfilter lässt alle Frequenzen unterhalb einer bestimmten Grenzfrequenz passieren und blockiert alle höheren Frequenzen vollständig.

Die Implementierung eines solchen Filters erfolgt in drei Schritten: Zunächst wird das Signal mittels FFT in den Frequenzbereich transformiert. Dann werden alle Frequenzkomponenten oberhalb der Grenzfrequenz auf null gesetzt. Schließlich wird das gefilterte Signal durch inverse FFT zurück in den Zeitbereich transformiert.

```python
# Beispiel: Tiefpassfilter durch Fourier-Transformation
def lowpass_filter(signal, cutoff_freq, sample_rate):
    """
    Wendet einen idealen Tiefpassfilter an.
    """
    # FFT
    F = np.fft.fft(signal)
    freqs = np.fft.fftfreq(len(signal), 1/sample_rate)

    # Filter anwenden
    F_filtered = F.copy()
    F_filtered[np.abs(freqs) > cutoff_freq] = 0

    # Rücktransformation
    signal_filtered = np.fft.ifft(F_filtered).real

    return signal_filtered
```

Diese Funktion nimmt ein Signal, eine Grenzfrequenz und die Abtastrate entgegen und liefert das gefilterte Signal zurück. Der Filter arbeitet durch einfaches Nullsetzen der unerwünschten Frequenzkomponenten. In praktischen Anwendungen verwendet man häufig sanftere Filter, die einen allmählichen Übergang zwischen Durchlass- und Sperrbereich aufweisen, um unerwünschte Artefakte zu vermeiden.

Diese Beispiele illustrieren die praktische Anwendung der Fourier-Methoden in der Signalverarbeitung. Die FFT bildet die Grundlage für zahlreiche Algorithmen in der digitalen Signalverarbeitung, der Bildverarbeitung und der Datenanalyse. Die Fähigkeit, Signale effizient zwischen Zeit- und Frequenzbereich zu transformieren, eröffnet vielfältige Möglichkeiten zur Analyse und Manipulation von Daten.