Übung 05: Spektrum einer Klaviersaite

Tobias Blesgen und Leonardo Thome

27.06.2021

Schlägt man eine Klaviersaite an, so verhält sie sich nach:

$$\frac{\partial^2\psi(x,t)}{\partial t^2} = c^2\frac{\partial^2\psi(x,t)}{\partial x^2} - \frac{\gamma}{l}|\frac{\partial\psi(x,t)}{\partial t}|(\frac{\partial\psi(x,t)}{\partial t}). \tag{1}$$

Wir wollen dieses Verhalten im Folgenden numerisch untersuchen. Hierzu schreiben wir Gleichung 1 mit $\Xi = \frac{x}{l}, \phi = \frac{\psi}{l}, \tau = \frac{tc}{l}$ um zu einer Dimensionslosen Gleichung:

$$\frac{\partial^2\phi(\Xi,\tau)}{\partial\tau^2} = \frac{\partial^2\phi(\Xi,\tau)}{\partial\Xi^2} - \gamma|\frac{\partial\phi(\Xi,\tau)}{\partial\tau}|(\frac{\phi(\Xi,\tau)}{\partial\tau}). \tag{2}$$

Um dieses Problem analytisch angehen zu können, müssen wir die Differentiale nähern und schreiben hierzu die Gleichung 2 um zu:

$$\frac{\psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j}}{\Delta\tau^2} = \frac{\psi_{i,j+1} - 2\psi_{i,j} + \psi_{i,j-1}}{\Delta\Xi^2} - \gamma|\frac{\psi_{i,j} - \psi_{i-1,j}}{\Delta\tau}|\frac{\psi_{i,j} - \psi_{i-1,j}}{\Delta\tau} \tag{3}$$

Dies lässt sich weiter umstellen zu:

$$\psi_{i+1,j} = \frac{\Delta\tau^2}{\Delta\Xi^2}(\psi_{i,j+1} - 2\psi_{i,j} + \psi_{i,j-1}) + 2\psi_{i,j} - \psi_{i-1,j} - \gamma|\psi_{i,j} - \psi_{i-1,j}|(\psi_{i,j} - \psi_{i-1,j}) \tag{4}$$

Diese Gleichung wird nun verwendet, um iterativ den nächsten Funktionswert an jedem Ort zu bestimmen. Als Randbedingung werden die Ränder, also $\Xi = 0$ und $\Xi = \Xi_{max}$ auf 0 gesetzt, da sie im Klavier befestigt sind.

Implementation des numerischen Verfahrens

Nach der Courant-Friedrichs-Lewy Stabilitätsbedingung müssen wir bei der Schrittweite Wahl beachten, dass stets

$$1 \geq c\frac{\Delta t}{\Delta x} \tag{5}$$

gilt. Setzen wir Ξ und τ ein, so erhalten wir nach einmaligen Umstellen die Ungleichung

$$\Delta\Xi \geq \frac{cl}{cl}\Delta\tau = \Delta\tau. \tag{6}$$

Wir müssen also darauf achten die τ-Schritte kleiner als die Ξ-Schritte zu halten.

```
#include<Rcpp.h>
#include<vector>
#include<algorithm>
#include<math.h>

using namespace Rcpp;


//[[Rcpp::export]]
NumericMatrix klaviersaite2(const double gamma, const int xSchritte,
                                                const double dt, const int
                                                zeitSchritte){
  // Array der Werte zur späteren Ausgabe
    NumericMatrix matrix(zeitSchritte, xSchritte);
  // Quelltext
    // Startwerte

    for (int i = 0; i<xSchritte; i++){
      matrix(0,i) = 0.0;
      matrix(1,i) = 0.0;
    }
    matrix(0,(int)(0.26*(xSchritte-1))) = 0.01/0.4;
    matrix(1,(int)(0.26*(xSchritte-1))) = 0.01/0.4;



    // Funktionsdurchläufe
    double dx = 1.0/(xSchritte - 1.0);
    double C = dt*dt/(dx*dx);

    for (int i = 2; i<zeitSchritte; i++){
      // Randbedingungen
      matrix(i,0) = 0.0;
      matrix(i,xSchritte-1) = 0.0;
      for (int j = 1; j<xSchritte-1; j++){
        matrix(i,j) = C*(matrix(i-1,j+1) - 2*matrix(i-1,j) +
          matrix(i-1,j-1)) + 2*matrix(i-1,j) - matrix(i-2,j) -
          gamma * fabs(matrix(i-1,j)-matrix(i-2,j)) *
          (matrix(i-1,j)-matrix(i-2,j));
      }
    }

  // Rückgabe für eine grafische Wiedergabe
    return matrix;
}
```

Stabilitätsüberprüfung

Wir wollen am Beispiel des $\gamma = 0$ Falls überprüfen ob unsere Methodik stabil ist. Wir würden erwarten dass die mittlere Amplitude über die Zeit konstant bleibt und tragen hierzu die Summe aller Funktionspunkte grafisch gegen die Zeit auf:

Wie wir sehen können, bleibt die Amplitudensumme über dem betrachten Bereich sehr konstant und wir

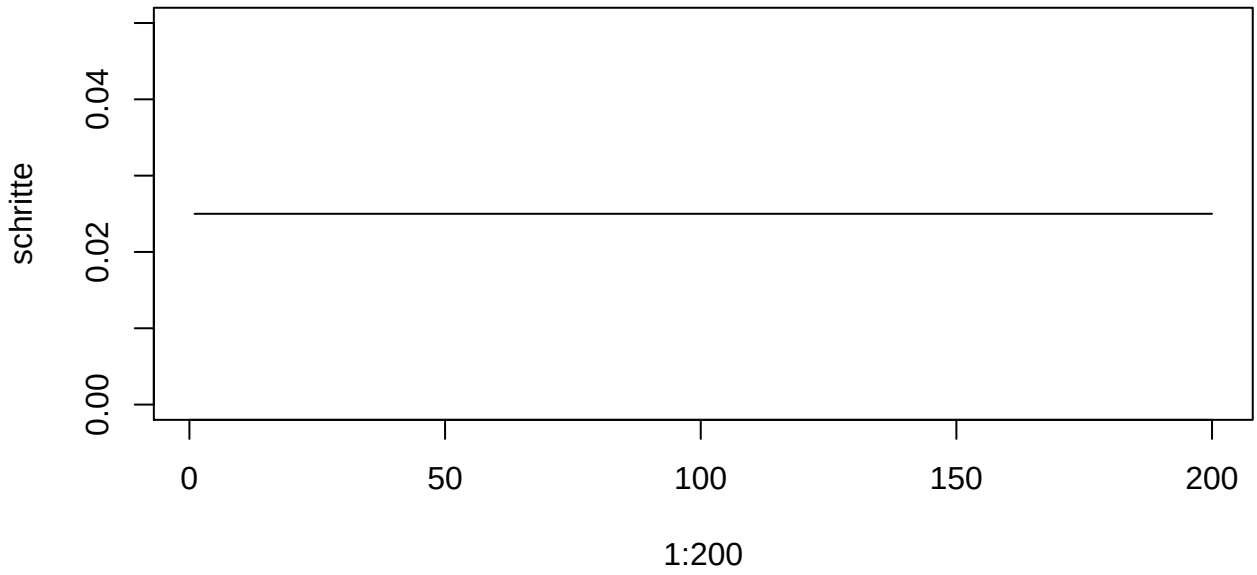

Abbildung 1: Stabilität bei keiner Abschwächung

können die Methode im folgenden auf ein gedämpftes System anwenden.

Anwendung

Wir wollen eine Klaviersaite schwingen lassen und auf seine Obertöne und den Grundton untersuchen. Betrachten wir in einer ersten Simulation die räumliche Schwingung zu verschiedenen Zeiten so können wir in Abb. **??** sehen wie sich die Schwingung ausbreitet ($\Delta\Xi = 2 \cdot 10^{-3}$ und $\Delta\tau = 10^{-}3$).

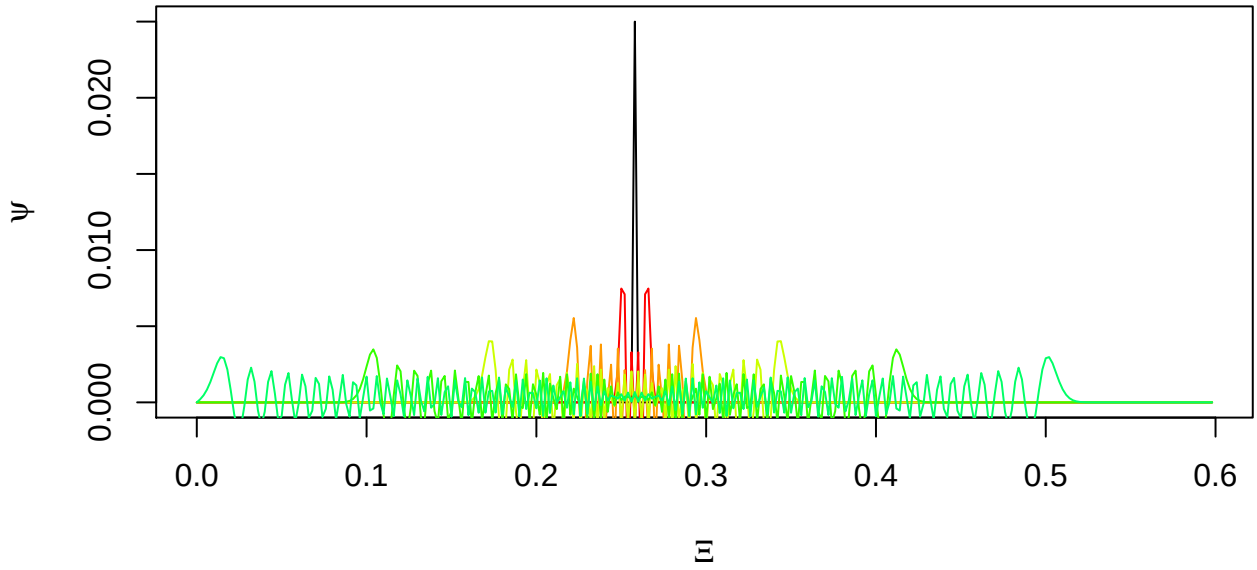

Abbildung 2: Räumlicher verlauf

Wir wollen im folgenden mit möglichst kleinen Diskretisierungslängen arbeiten, ohne die Rechenzeiten extrem in die Länge zu ziehen und haben uns für $\Delta\tau = 10^{-}4$ und $\Delta\Xi = \frac{4}{3} \cdot 10^{-3}$ entschieden. Diese Längen halten auch die geforderte Bedingung Gl. 6 ein.

Zeitlicher Verlauf

Für die ganauere Untersuchung benötigen die zeitliche Entwicklung an einem Ort. Wir haben in Abb. 3 den Verlauf an der Stelle $\Xi = 0,3$ gewählt und für die verschiedenen Zeiten aufgetragen. Die Kurve beginnt in der Ruhelage bis das erste Hochpunkt den Ort erreicht. Es lässt sich nun eine exponentiell abfallende Sinuskurve beobachten, bis die am Rand reflektierte Welle den Punkt erreicht. Die verbleibende Überlagerung stabilisiert sich zu einer annähernd gleichmässigen Schwebung.

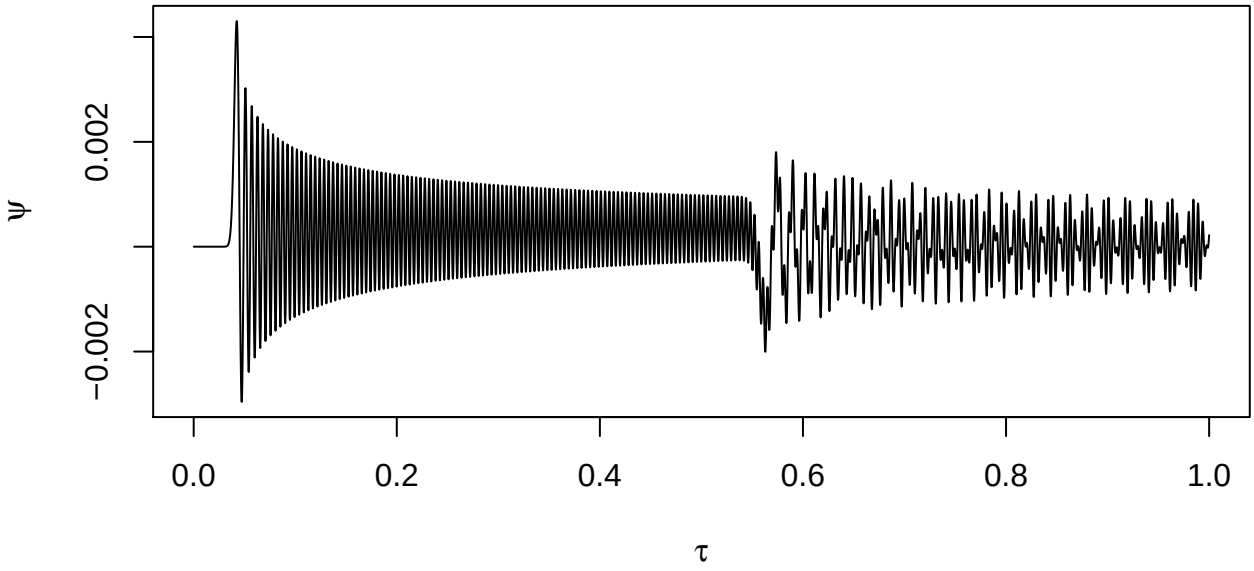

Abbildung 3: Zeitliche Entwicklung

Fast Fourier Transform

Mit dem Ziel die Schwingung in ihre Frequenzen aufzuteilen, werden wir die Werte der zeitlichen Entwicklung Fourier transformieren. Diese Transformation zerlegt eine reele Funktion in eine Summe von gewichteten Cosinusfunktionen. Wir werden den Fast Fourier Transform-Algorithmus verwenden, da wir mit diskreten Werten arbeiten. Die n erhaltenen Werte entsprechen anschließend den Gewichtungen der Cosini in aufsteigender Wertigkeit, also:

$$\mathrm{T}_f(x) = \frac{1}{\sqrt{2\pi}} \sum_k = 1^n g_k \cos(kx) \tag{7}$$

```
#include <Rcpp.h>
#include <stdlib.h>

using namespace Rcpp;

double pi = 3.14159;
std::complex<double> I = 1i;

// Generiere die w Faktoren
std::vector<double> wInit(int n){
  std::vector<double> W(n);
  for (int a = 0; a < n; a++)
```

```
    {
        W[a]= cos((-2.*pi*(double)a)/((double)n));
    }
  return W;
}

// Funktion zum Umsortieren der verdrehten Terme
std::vector<int> fftResort(const int n){
    std::vector<int> k(n);
    int l = 0, m;
    for (int i = 0; i <= n-2; i++)
    {
        k[i] = l;
        m = n/2;
        while (m <= l)
        {
            l -= m;
            m /= 2;
        }
        l += m;
    }
    k[n-1] = n-1;
    return k;
}

// Fast Fourier Transform - Funktion (nimmt die z_i und gibt die g_i unsortiert aus)
//[[Rcpp::export]]
std::vector<double> fft2 (std::vector<double> z, const int r){
    const int n = (int)pow(2,r);
    int m = n/2;
    int K = 1;
    std::vector<double> w = wInit(n);
    int a, b;

    for (int i = 0; i < r; i++)
    {
        for (int k = 0; k < K; k++)
        {
            for (int j = 0; j < m; j++)
            {
                a = 2*k*m +j;
                b = a + m;
                z[a] += z[b];
                z[b] = w[((K*j) % n)]*(z[a] - 2.0*z[b]);
            }
        }
        m /= 2;
        K *= 2;
    }
    std::vector<int> index = fftResort(n);
    std::vector<double> ausgabe(n);
    for (int i = 0; i < n; i++)
    {
```

```
        ausgabe[i]= z[index[i]] / sqrt(n);
    }
    return ausgabe;
}
```

Zerlegen wir nun verschiedene Orte nach der Fouriertransformation, so erhalten wir Abb. 4.

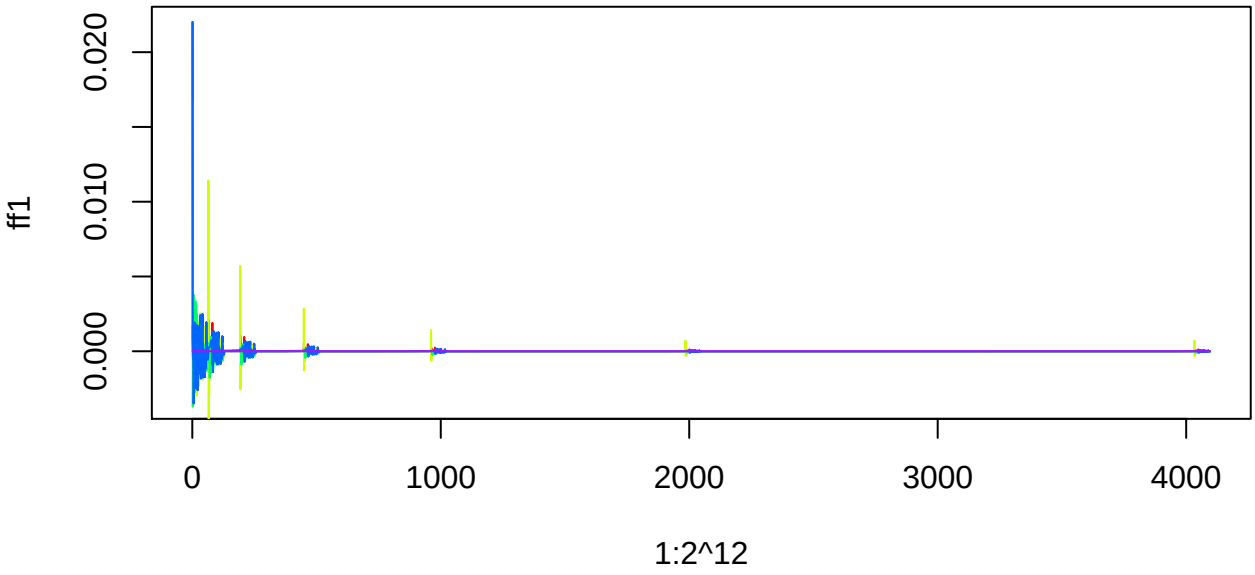

Abbildung 4: Fourier zerlegtes Spektrum

Fazit

Wir erhalten bei den meisten Werten ein nicht chaotisches Verhalten zwischen θ und $\dot{\theta}$, können jedoch auch chaotische Situationen konstruieren. Dies ist im physikalischen Sinne auch nachvollziehbar, da die Eigenrotation des Mondes durch geschickt gewählte Bahnen stark von der Radiusvariation beeinflusst werden kann.

Wir können aber sehen, dass dieses Verhalten nicht immer eintritt und wir im Allgemeinen ein strukturiertes Verhalten vorfinden.

Literatur

[1] Keplersche Gesetze https://de.wikipedia.org/wiki/Keplersche_Gesetze, Stand 22.06.2021