Migration Seismischer Daten

Aufgabe 2

Nun gilt es, synthetische Seismogramme zu migrieren.
Dazu schreiben wir ein Modul in *Pyhton*, in welchem wir die benötigten Prozeduren als Funktionen implementieren.
Dieses Modul laden wir im interaktiven Pyhton-Interpreter:

```
>>> import kirchhoffmigration as k
```

Nun können wir die Daten darstellen (Abb. 1):

```
>>> k.PlotImg(k.data,'unprocessed')
```

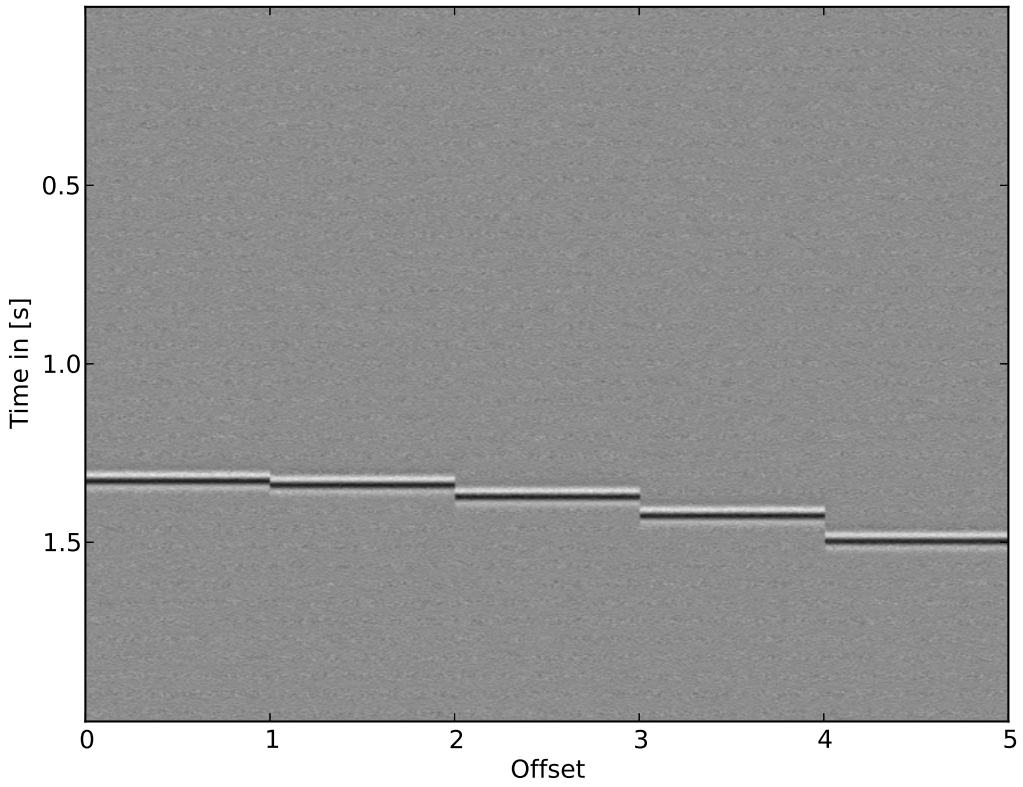

Abb. 1: Unprozessierte Daten

a. - Frequenzgehalt der Daten

Um den Frequenzgehalt der Daten abschätzen zu können, verwenden wir die *fast Fourrier Transformation* aus dem *SciPy*-Modul[1].

```
>>> k.PlotSpectrum(1)
```

In Abbildung 2 ist zu erkennen, dass die Hauptfrequenz des Signals bei ca. 25 Hertz liegt. Damit liegt sie deutlich unter der Nyquist Frequenz von 250 Hertz. Die Abtastfrequenz hätte also noch niedriger gewählt werden können, um Datenvolumen und damit Speicherplatz und Rechenzeit zu sparen.

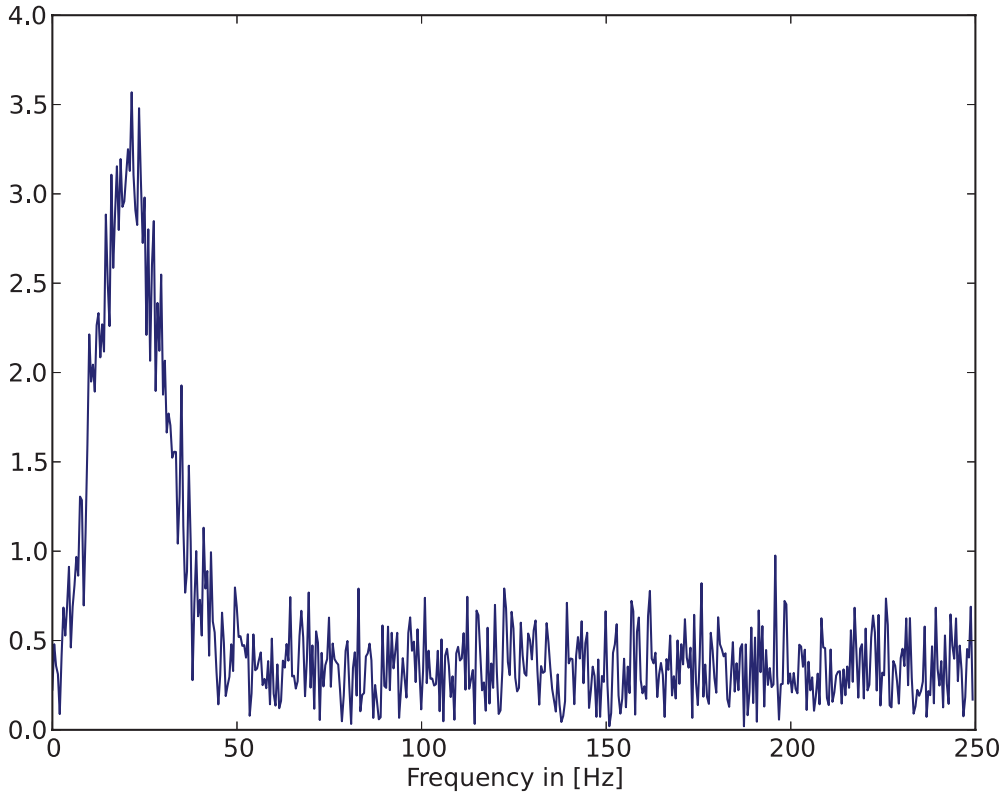

Abb. 2: Spektrum der 2. Spur der Originaldaten

b. - Migrationsverfahren

Da die Daten als *Constant Offset Gather* vorliegen, bietet sich eine *Kirchhoff Inversion* an.
Hierbei diskretisieren wir zunächst den Untergrund und summieren dann für jeden Untergrundpunkt die dazugehörigen (auf seiner virtuellen Diffraktionshyperbel liegenden) Reflektivitäten aus den Daten auf.

[1] http://www.scipy.org

Der Vorteil dieser Methode ist, dass wir sowohl das Abbild als auch die Reflexionsamplituden rekonstruieren können (*True Amplitude Migration*. Dadurch lassen sich möglicherweise Rückschlüsse auf die Schereigenschaften (und damit Porosität und Permeabilität) des Materials ziehen.

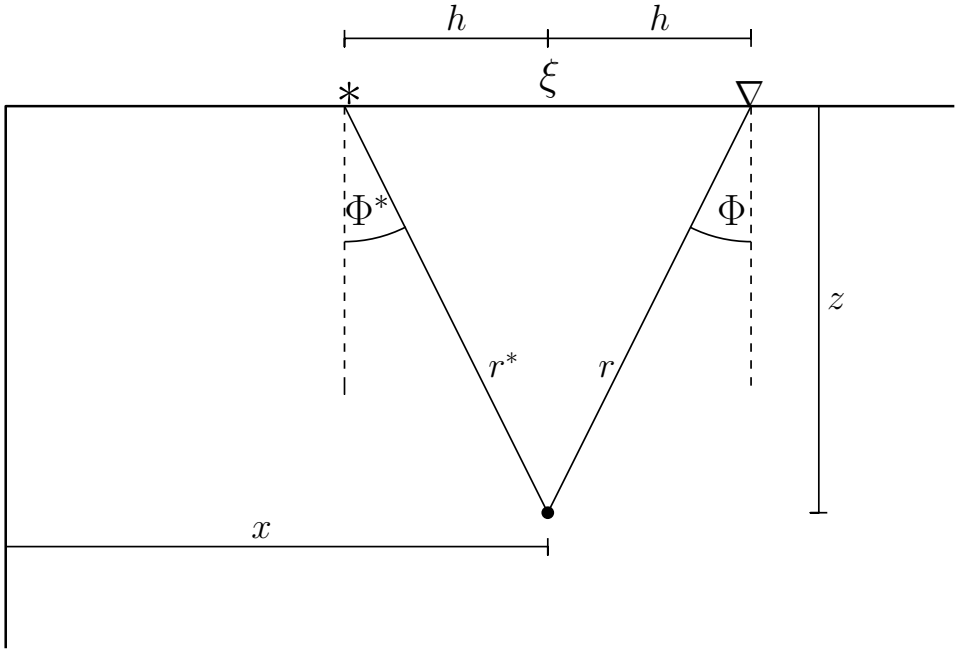

Abb. 3: Geometrie der Kirchhoff Inversion

Aus dem Skript wissen wir:

$$R(x,z) = \int_{-\infty}^{\infty} d\xi \quad u_F(\xi, z=0, \frac{r^* + r}{v}) \quad \omega_{CO} \frac{1}{\sqrt{2\pi}} \tag{1}$$

mit

$$\omega_{CO} = \frac{1}{v} \quad (\cos\Phi \sqrt{\frac{r^*}{r}} + \cos\Phi^* \sqrt{\frac{r}{r^*}}) \tag{2}$$

Aus geometrischen Überlegungen in Abbildung 3 erhalten wir:

$$r^* = \sqrt{(x - (\xi - h))^2 + z^2} \tag{3}$$

$$r = \sqrt{(x - (\xi + h))^2 + z^2} \tag{4}$$

und

$$\Phi^* = \arccos(\frac{z}{r^*}) \tag{5}$$

$$\Phi = \arccos(\frac{z}{r}) \tag{6}$$

c. - Diskretisierung des Untergrundes

Für die vertikale Diskretisierung ist die Wellenlänge des Signals von Bedeutung. Es muss gelten:

$$\Delta z < \frac{\lambda}{2} \tag{7}$$

In der Praxis wählt man allerdings meistens eine feinere Diskretisierung von $\Delta z < \frac{\lambda}{8}$. Wir erhalten, bei einer Geschwindigkeit von $3000\frac{m}{s}$, eine Wellenlänge von 120 Metern und wählen $\Delta z = 10m$.

Lateral sollte Δx in etwa dem Spurabstand (in unserem Fall 20m) entsprechen.

f./g. - Geschwindigkeits- Tiefenanalyse

Um die korrekte Untergrundgeschwindigkeit zu bestimmen, machen wir uns zunutze, dass die Migration mit der korrekten Geschwindigkeit die Reflexion auf einen Punkt fokussiert, wohingegen eine falsche Geschwindigkeit zum 'verschmieren' der Energie führt.

Wir können also eine Geschwindigkeitsanalyse durchführen, indem wir für ein einzelnes festes x mit verschiedenen Geschwindigkeiten migrieren und die Ergebnisse darstellen (Abb. 4). Dies funktioniert allerdings nur, weil wir eine Linienquelle haben.

```
>>> k.v_analysis(2000, 5000)
```

Damit erhalten wir eine Geschwindigkeit von $v = 2950\frac{m}{s}$ und eine Reflektortiefe von $z = 1970m$.
Wir können somit den theoretischen Zero-Offset Einsatz im Seismogramm bestimmen:

$$t = \frac{2z}{v} = \frac{2 \cdot 1970m}{2950\frac{m}{s}} \approx 1.336s \tag{8}$$

Dieses Ergebnis passt sehr gut zum beobachtbaren Einsatz im Seismogramm.

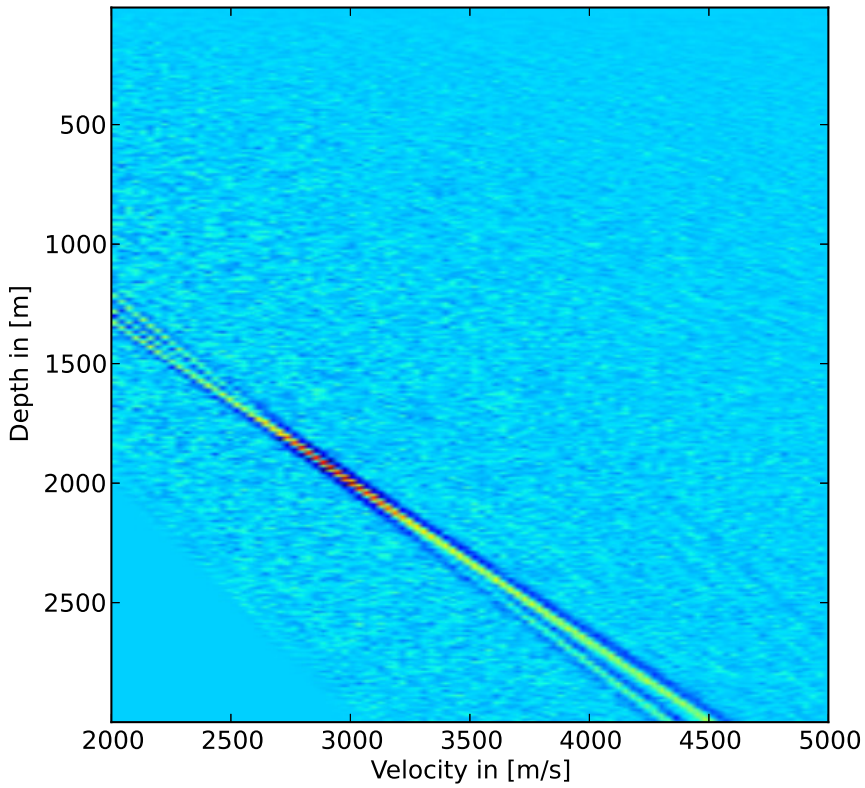

Abb. 4: Geschwindigkeits- Tiefenanalyse

d. - Migrierte Sektion

```
>>> k.full_migration(data)
```

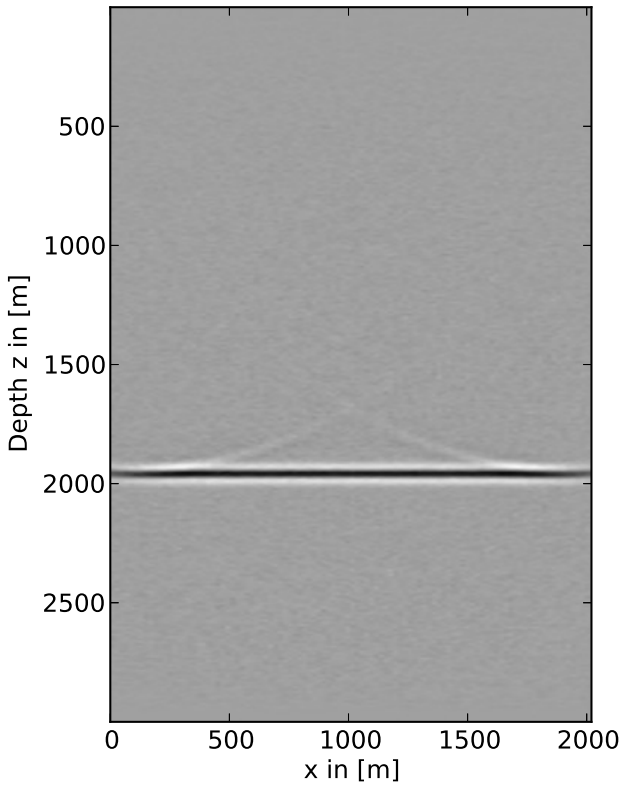

Abb. 5: Migrierte Sektion mit $v = 2950\frac{m}{s}$

In der migrierten Sektion ist der Reflektor in einer Tiefe von 1970 Metern abgebildet. Da es sich um eine Linienquelle handelt, lässt sich nicht erkennen, ob Diffraktionen durch die Migration kollabieren.
Von den Rändern ausgehend sind Artefakte zu erkennen, die durch das abrupte Ende der Spuren zustande kommen.

e. - Amplitudenverläufe

```
>>> result = k.full_migration(data)
>>> k.check_amplitudes(result)
```

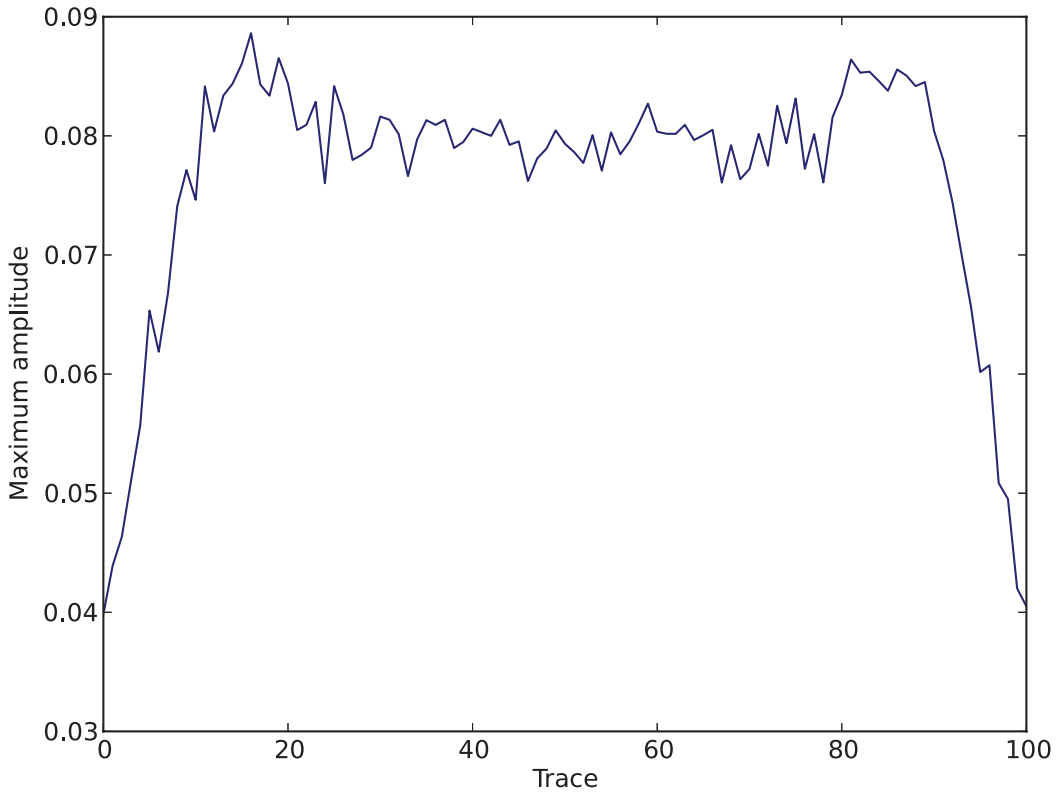

Abb. 6: Maximalamplitude in Abhängigkeit der Spur

In Abbildung 6 ist zu erkennen, dass die Amplituden zum Rand hin abnehmen. Dies dadurch zu erklären, dass an den Rändern nur entlang der einen Hälfte der Diffraktionshyperbel entlang summiert wird. Auf der anderen Seite befinden sich keine Daten mehr.
Interessant sind die Maxima, die auftreten bevor die Amplitude abfällt. Wir vermuten, dass dieses Verhalten mit den Randartefarkten zusammenhängt.

h. - Bewertung

i. - Verbesserungsvorschläge

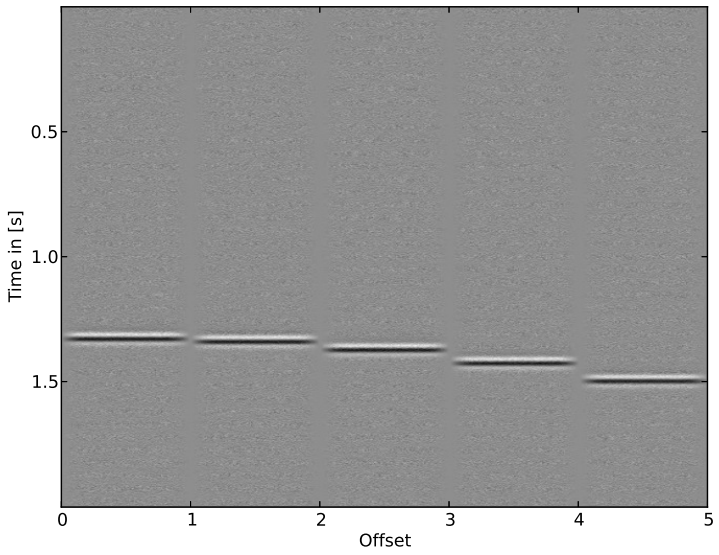

Abb. 7: Daten mit einem Taper von 20 Traces

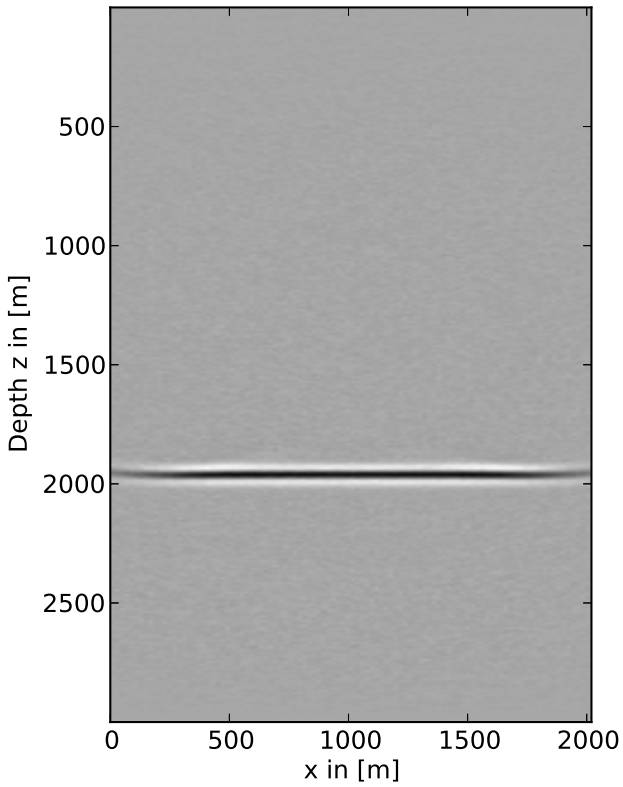

Abb. 8: Migrierte Sektion der vorher mit Taper versehenen Daten

Optimierung des Codes in Cython

Da Python eine interpretierte Sprache ist und dementsprechend langsam, haben wir den Migrationsalgorithmus noch einmal in *Cython*[2] implementiert.

```
>>> k.benchmark()
Benchmarking python vs cython:
Migrating 10 times with nx= 1
python code:
83.7969629765
cython code:
39.1083641052
```

Durch die statische Typisierung der Variablen ist der Code etwa doppelt so schnell geworden, ohne dass dadurch die Lesbarkeit beeinträchtigt wurde.

[2] http://cython.org