Abgabe 1

April 22, 2018

1 Übungsblatt 1 (Sprach und Signalverarbeitung)

Für diese Aufgabe verwenden wir die Pakete `WAV` um Klangdateien auszulesen und `Plots` für das Plotten. Zudem verwenden wir plotly für interaktive Plots (bzw. die Python library `matplotlib` für statische Grafiken).

```
In [15]: #Pkg.add("WAV")
         #Pkg.add("Plots")
         using WAV,Plots

         #plotly()  ## Für interaktive Plots
         pyplot()  ## Für pdf-Export
```

```
Out[15]: Plots.PyPlotBackend()
```

Im Folgenden arbeiten wir mit zwei Klangdateien.

```
In [16]: (s1,fs1) = wavread("../speech1.wav");
         (s2,fs2) = wavread("../speech2.wav");
```

1.1 Aufgabe 1

1.1.1 (a)

Die Abtastrate ist der zweite Ausgabeparameter der Funktion `wavread`

```
In [17]: println("Abtastrate für speech1.wav: ",fs1," Hz")
         println("Abtastrate für speech2.wav: ",fs2," Hz")
```

```
Abtastrate für speech1.wav: 16000.0 Hz
Abtastrate für speech2.wav: 16000.0 Hz
```

1.1.2 (b)

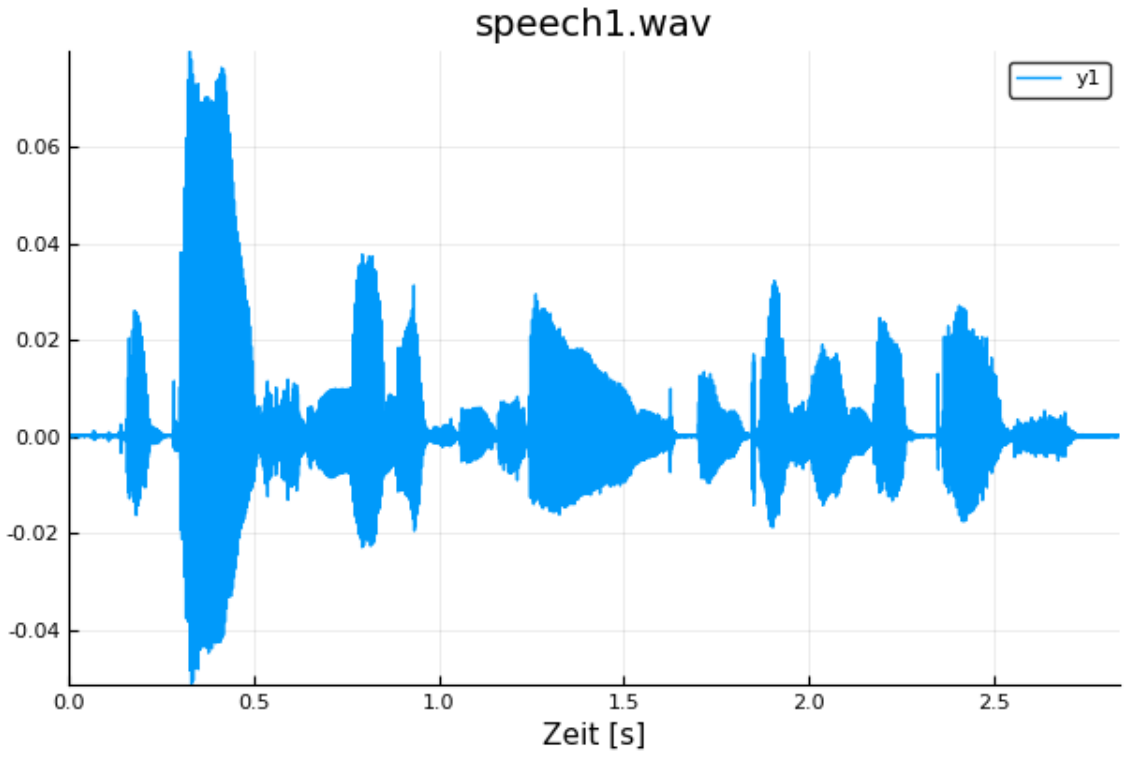
Ist `s` ein Signal mit Abtastfrequenz `fs`, so findet die erste Messung zum Zeitpunkt 0, die zweite zum Zeitpunkt `period_length` und im Allgemeinen die `i`-te Messung zum Zeitpunkt `(i-1)*period_length` statt, wobei `period_length = 1/fs` die Inverse der Frequenz ist.

(Bemerke: Die Indizierung von Vektoren beginnt sowohl in Julia als auch in Matlab bei 1)

```
In [18]: v_sample_time1 = (0:(length(s1)-1))/fs1;
         v_sample_time2 = (0:(length(s2)-1))/fs2;
```

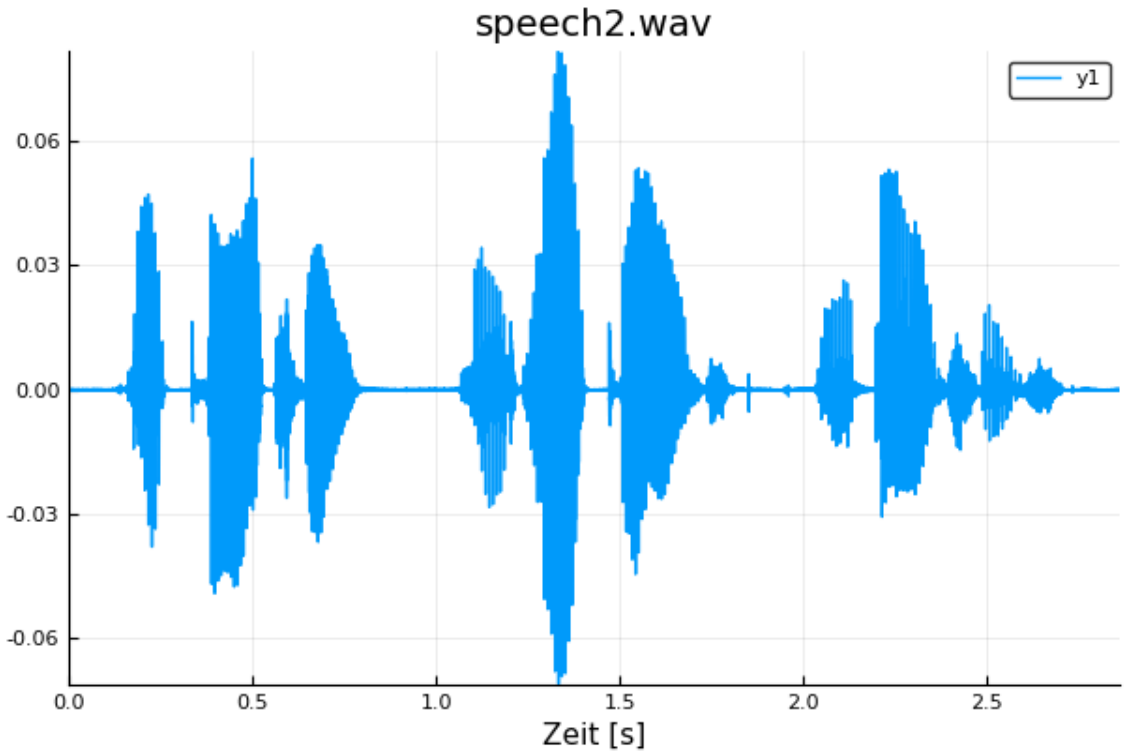
```
In [19]: plot(v_sample_time1,s1;title="speech1.wav",xlab="Zeit [s]")
```

Out[19]:




```
In [20]: plot(v_sample_time2,s2;title="speech2.wav",xlab="Zeit [s]")
```

Out[20]:



- stille Regionen
  - zeichnen sich durch eine sehr geringe Amplitude aus
- stimmhafte Regionen
  - bestehen aus einem periodischen Anteil multipliziert mit einer variierenden Amplitude
  - die Änderung der Amplitude ist bei Sprachsignalen (im Gegensatz zum Gesang) buckelförmig und findet auf einer grösSeren Skala statt (10-20 Perioden pro Buckel)
  - relativ hohe Amplitude
- stimmlos Regionen
  - Hohe UnregelmäSSigkeit (bei jeder Skalierung)
  - Amplitudenmaxima variieren mehr (keine Buckel)

1.1.3 (c)

Schätzung für Signal 1 - Periode von $t_0 = 0.3428125$ bis $t_1 = 0.3478125$ - Frequenz: $\frac{1}{t_1 - t_0} \approx 200Hz$
Schätzung für Signal 2 - Periode von $t_0 = 0.6850625$ bis $t_1 = 0.693$ - Frequenz: $\frac{1}{t_1 - t_0} \approx 126Hz$
Rechnung:

```
In [21]: freqFromPeriod = function(von, bis) 1/(bis-von) end
         println(freqFromPeriod(0.3428125, 0.3478125))
         println(freqFromPeriod(0.6850625,0.693))
```

```
199.99999999999983
125.98425196850503
```

Anmerkung: Um stabilere Schätzungen zu erreichen hätten wir auch die Länge von n Perioden messen können um eine durchschnittliche Periodenlänge zu bestimmen.

1.2 Aufgabe 2

Gegeben sind - ein Signal `v_signal` - seine Abtastrate `sampling_rate` - die Länge eines Zeitfensters `frame_length` - Abstand zwischen den Startpunkten zweier aufeinanderfolgender Zeitfenster `frame_shift` (uff, das geht sicher eloquenter)

Die erste Aufgabe besteht darin die Abmessungen für unsere Zeitfenster, die in Sekunden angegeben sind, als eine Anzahl von Samples auszudrücken. Dadurch ergeben sich folgende Definitionen:

- `samples_per_frame = Int(floor(frame_length * sampling_rate))`
- `samples_per_shift = Int(floor(frame_shift * sampling_rate))`

Als nächstes stellt sich die Frage wie viele Fenster in unser Signal passen. Wir nehmen sinnvollerweise an, dass in unser Signal mindestens ein Fenster (genauer gesagt, das erste Fenster) passt. Jedes darauf folgende Fenster *konsumiert* von den restlichen `length(v_signal) - samples_per_frame` Samples genau `samples_per_shift`. Dementsprechend ergibt sich die Formel - `shift_count = 1 + Int(floor((length(v_signal) - samples_per_frame)/samples_per_shift))`

Diese Daten reichen nun aus, um die Ausgabematrix zu befüllen.

Für die Mittelpunkte der Zeitfenster ergibt sich durch folgende Überlegung: Der Mittelpunkt des ersten Fensters befindet sich bei `frame_length/2`. Jeder darauffolgende Mittelpunkt ergibt sich aus einer Verschiebung um `frame_shift`. - `v_time_frame = frame_length/2 + frame_shift * (0:(shift_count-1))`

```
In [22]: my_windowing = function(v_signal, sampling_rate, frame_length, frame_shift)
             samples_per_frame = Int(floor(frame_length * sampling_rate))
             samples_per_shift = Int(floor(frame_shift * sampling_rate))
             shift_count = Int(floor((length(v_signal) - (samples_per_frame - samples_per_shift
             m_frames = zeros(samples_per_shift, shift_count)
             for j in 1:shift_count
                 for i in 1:samples_per_shift
                     m_frames[i,j] = v_signal[samples_per_shift*(j-1)+i]
                 end
             end
             v_time_frame = frame_length/2 + frame_shift * (0:(shift_count-1))

             (m_frames, v_time_frame)
         end
```

```
Out[22]: (::#14) (generic function with 1 method)
```

1.3 Aufgabe 3

1.3.1 (a)

```
In [23]: (frames1,frame_times1) = my_windowing(s1,fs1,32e-3,16e-3);
         (frames2,frame_times2) = my_windowing(s2,fs2,32e-3,16e-3);
```

1.3.2 (b)

```
In [24]: autocor = x -> conv(x,reverse(x))/length(x)
```

```
Out[24]: (::#16) (generic function with 1 method)
```

1.3.3 (c)

Trunkierte version von `autocor`, bei der wir nur positive Shifts erlauben.

```
In [25]: autocor_t = x -> autocor(x)[length(x):end]
```

```
Out[25]: (::#18) (generic function with 1 method)
```

1.3.4 (d)

```
In [26]: ff_estimator = function(s,fs;frame_length=32e-3, frame_shift=16e-3)
             (frames, frame_times) = my_windowing(s,fs,frame_length,frame_shift)
             (samples_per_frame, shift_count) = size(frames)

             ac = zeros(frames)      # auto correlations
             freq = zeros(frame_times) # ff estimate per frame

             # 80Hz:400Hz frequency window
             lb = Int(floor(fs/400))  # lower bound
             ub = Int(floor(fs/80))   # upper bound

             for j in 1:shift_count
                 ac[:,j] = autocor_t(frames[:,j])
                 period = lb - 1 + indmax(ac[lb:ub,j])   # period estimate (measured in number
                 freq[j] = fs/period
             end

             return (frame_times, freq)
         end
```

```
Out[26]: (::#20) (generic function with 1 method)
```

1.3.5 (e)

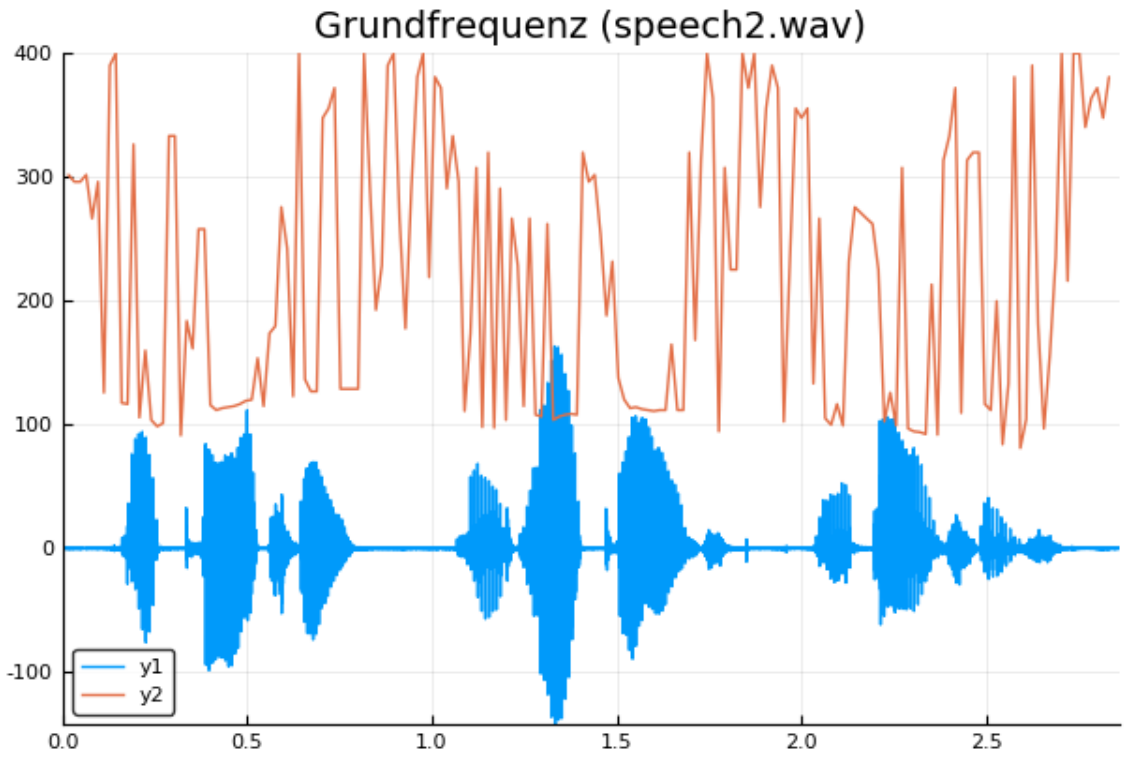
Wir wollen unsere Schätzung `ff` für die Grundfrequenz und zum Vergleich dazu die Signalamplitude `s` im gleichen Plot darstellen. Da die Einheiten, geschweige denn die GröSSenordnungen dieser Datenreihen sehr unterschiedlich sind, multiplizieren wir zwecks Darstellung die Amplitude um 2000.

In [27]:
```
(tf1,ff1) = ff_estimator(s1,fs1);
plot([v_sample_time1,tf1],[s1*2000,ff1],title="Grundfrequenz (speech1.wav)")
```

Out[27]:

In [28]:
```
(tf2,ff2) = ff_estimator(s2,fs2);
plot([v_sample_time2,tf2],[s2*2000,ff2],title="Grundfrequenz (speech2.wav)")
```

Out[28]:



Bemerkungen

- Der Grundfrequenzschätzer produziert für stimmlose oder stille Regionen stark fluktuierende Werte. Das entspricht unserer Erwartung, dass stimmlose Regionen einem weiSSen Rauschen ähneln und wir im Frequenzbereich keine markanten Peaks finden.
- Stimmhafte Regionen sind in der Regel flach.
  - Dies ist jedoch nicht immer der Fall: Für den Laut bei 1.1s bis 1.2s (speech2.wav) oder bei 2.2s bis 2.3s (speech2.wav) schätzt der Algorithmus zwischen 100Hz und 300Hz.
  - Für stimmhafte Regionen scheint der Grundfrequenzschätzer bei hohen Stimmen zuverlässiger zu arbeiten als bei tiefen stimmen.
- Die manuell geschätzen Frequenzwerte (200Hz, 126Hz) stimmen einigermaSSen gut mit den automatisch geschätzen Werten (213Hz, 127Hz) überein.

Weitere Bemerkungen

- Eine Schwäche der Schätzung mittels Autokorrelation ist neben der Peakbestimmung auch die Länge und Positionierung der Zeitfenster auf welche die Autokorrelation angewendet wird.
- Man könnte sich überlegen zunächst die Zeitbereiche in “stimmhaft, stimmlos, still” zu kategorisieren. Dies würde es uns erlauben insbesondere die stimmlosen, rauschigen Regionen nicht in die Schätzung einzubeziehen.
- Weiters sind Laute nicht rein periodisch, sondern haben einen impulsartigen, buckeligen Anteil. Eine Normalisierung der Amplitude könnte die Schätzungen verbessern.