**Time-Frequency Analysis of One-Dimensional Audio Data**

Maxwell Weil

**Abstract**

In this homework, we will be examining techniques for representing one-dimensional data in both the time and frequency domains simultaneously. We will first overview of the concepts and strategies used in this investigation, providing a theoretical background for this paper. Next, we will explore our data in both the time and frequency domains using MATLAB software coupled with signal and wavelet analyzer toolboxes. Ultimately, a better understanding and appreciation for the computational methods to visualize and analyze time-frequency data will be established.

**I. Introduction**

The analysis of signals in the time and frequency domains is inherently problematic. To view a signal in the time domain, we lose the ability to see frequencies. Similarly, in the frequency domain, we can’t distinguish when certain information is occurring. This issue limits the ability to fully analyze and breakdown data. To combat this problem, a mathematical conversion called the Gabor transform was created. This technique computes a Fourier transform over small periods of time, allowing preservation of both time and frequency information. However, there are limits to the effectiveness of the Gabor transform.

In this work, the Gabor transform will be used on several datasets and analyzed in its entirety. Additionally, related methods of time-frequency analysis will be demonstrated and compared to this transform. Using these strategies, we will process one-dimensional sound data taken from two different instruments. In doing this, we aim to visualize how this data is changing in both the time and frequency domains. The differences and similarities between the sound samples will be compared, granting a comprehensive understanding of the data. Before these analyses, a we will numerically explore the algorithms and methods that we will use.

**II. Theoretical Background**

The Gabor transform is the centerpiece of this homework, so it is necessary to explain its fundamentals. As mentioned above, the Gabor transform computes the Fourier transform of a signal over various periods of time. Mathematically, the Gabor transform is shown in equation 1 below. Note that all equations have been adapted from [1].

EQ. 1

Looking at this formula, there is a clear resemblance to the Fourier transform equation. However, the most notable differences are the variables ω and τ substituting for k and t, respectively, and the inclusion of a new function, g(τ-t). ω is essentially the same as k, although the units have changed from 1/seconds to radians/seconds. τ is now the time, while t represents a shift. This is better understood in the context of g(τ-t), which is a window centered at time t (shifted along τ). In the Gabor transform, g(τ-t) take the shape of a Gaussian window, as outlined in equation 2.

EQ. 2

In this equation, a dictates the width of the window, and is inversely proportionally to the window size. This window is then multiplied by the signal of interest in the time domain, x(τ), and then transformed using a Fourier transform. The window shifts along the time signal, generating many Fourier transforms that make up what is called a spectrogram. Essentially, a spectrogram is just a representation of many Fourier transforms over different time windows. The Gabor transform that creates the spectrogram is also known as the short-time Fourier transform (STFT) for this reason. As with the Fourier transform, the STFT is invertible, with the inverse shown in equation 3.

EQ. 3

The inverse transform is again similar to the inverse of the Fourier transform. Note that a double integral is used instead, as integration is over all frequencies and all time. However, in order to computationally utilize the STFT, we also have to have a discretized form, which can be seen in equation 4.

EQ. 4

In this form, both frequency and time have been discretized into m and n, respectively. This results in a basic change of variables throughout the equation. This equation will form the basis of our work in this assignment. Additionally, we will be using different types of windows within the STFT. In addition to the typical Gaussian window, a Shannon window and Mexican hat window will be analyzed. The equations for the Shannon and Mexican hat windows are also given in equations 5 and 6, respectively.

EQ. 5

EQ. 6

A screenshot of a cell phone

Description automatically generated

*Figure 1. Plots of various window type for use in the Gabor transform. The first column shows each window individual, the seconds shows the window multiplied by the investigated audio signal, “Handel,” and the third shows the resulting frequency domain of the signal. Each row looks at a different window style, including the traditional Gabor (gaussian), Shannon and Mexican Hat windows. Note that the Gabor transform essentially functions as many Fourier transforms of a filtered time signal, with the filter moving along in time steps.*

*Figure 2. Spectrograms produced by using various window sizes and time steps with the Gabor transform are shown. The traditional Gabor (gaussian) window is used, the equation for which is shown in equation 2. In the left most four plots a time step of 0.1 seconds was used, while a values varied from 1 to 10000. In the the right most four plots, an a value of 100 was kept constant while the time step size changed from 0.05 seconds to 1 second.*

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

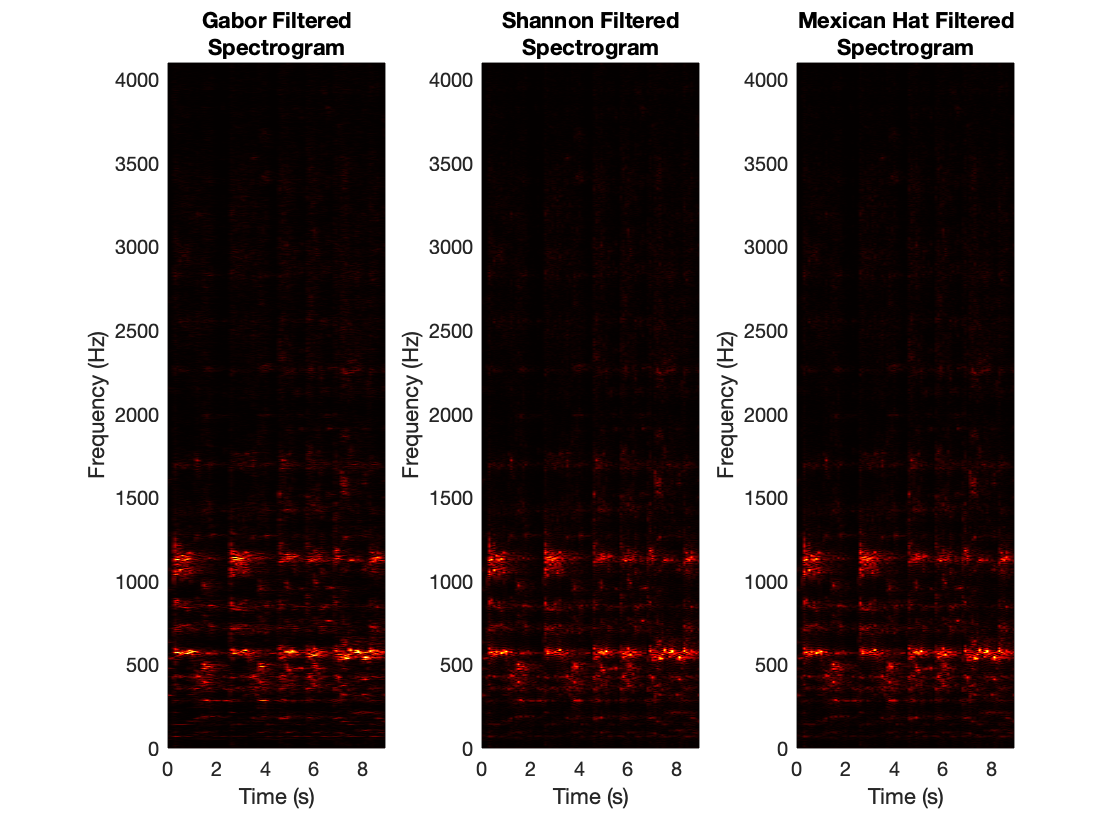
The Shannon window, s(τ-t), makes use of the Heaviside function, H, where negative values are set to 0, and positive values are set to 1. This window ends up appearing as a square pulse of width w. The Mexican hat window is similar to the Gabor window, though it aims to capture additional low frequencies by including additional oscillations. Examples of these three windows and their impact on filtering in time are shown in Figure 1.

Using these different windows, we will be aiming to computing various Gabor transforms on a few one-dimensional audio signals. However, the Gabor transform has its limitations. Namely, due to the uncertainty relationship between the time and the frequency domain, the resolution of this transform is limited. Filtering more precisely in time will give us finer time resolution, but we will lose the ability to see longer waveforms representing low frequencies. Likewise, in order to capture low frequencies, the time window must be large, removing our ability to discern precise time points. When the window is expanded infinitely, the Gabor transform actually reverts to the Fourier transform, due to losing all time resolution. Related to this issue, is the size of time steps. Small time steps result in oversampling, where data becomes choppy and challenging to relate to adjacent points. Very large time steps blur together frequencies, making it harder to tell when different frequencies are happening in time. Because of these issues, appropriate window sizes and time steps must be selected before performing time-frequency analysis. Figure 2 explores these concepts visually, by comparing the spectrograms of Gabor transforms with various windows widths and time steps.

**III. Algorithm Implementation and Development**

In this paper, several aspects of the Gabor transform are explored through the use of MATLAB software. First, windows used were visualized in the time domain, as shown in Figure 1 (lines 25-30). These windows were multiplied by an audio signal of interest, Handel, and illustrated in both time and frequency domains (lines 37-103). Note that the frequencies used throughout this work were scaled by a factor of 1/(sampling rate). This is because we wished to view the frequencies in Hz, rather than in radians per second as done previously.

*Figure 3. Resulting spectrograms for Gabor transforms computed with various window types. Time steps for the transform were all 0.1 seconds, while window widths varied. Both the traditional Gabor window and the Mexican hat window used an a = 100, while the Shannon window used a window width of 0.1 seconds.*

To compute the transform, and audio signal Handel was stepped through in time, multiplied by a window centered about that time point, and then transformed into the frequency domain (lines 123-129). The resulting frequency information was stored in a matrix, with rows representing the frequency data and columns containing the time point (line 132). This matrix was then visualized as a spectrogram, using the pcolor command in MATLAB (lines 136-143). This process will be used several times throughout this investigation.

As shown in Figure 2, components of the Gabor transform were inspected, and it was found that a gaussian window with a = 100 and a transform with a time step of 0.1 seconds were most optimal for our purposes. Following this, the gaussian, Shannon, and Mexican hat windows were compared, again using the Handel audio signal. The results for this can be seen in Figure 3. The same process as before was followed, though for different window equations.

After these initial analyses, two additional audio samples were analyzed using optimized methods. Each audio signal was a recording of “Mary Had a Little Lamb,” though played on different instruments (a piano and a recorder). These signals were read into MATLAB and downsampled to avoid computational difficulties (lines 266-270). Then, they were transformed into spectrograms using the Gabor transformation with a gaussian window (lines 296-300). Additionally, the maximal frequency at each time step was filtered (lines 303-311) using a gaussian window with a = 0.01. This was done to avoid overtones interfering with the reproduction of music scores for each instrument (this will be explained further in the results). The raw spectrograms without overtone filter can be seen in Figure 4 (lines 318-325).

Spectrograms with filtered overtones were generated and additional steps were taken to extract the music score for both the piano and recorder audio signals from these. To do this, maximal frequencies were processed by finding values over a certain threshold (relating to a note being played), separated by gaps of relative silence (releasing of a note, lines 329-335). These tones were then transposed into alphabetical notes, and the resulting music score was recreated, as shown in Figure 5.

*Figure 4. Raw spectrograms of a piano and recorder playing “Mary Had a Little Lamb.” A traditional Gabor filter was used with an a value of 100. Note the differences between each recording. While the recorder clearly plays at a higher frequency, different overtones can also be seen. If a note is played at a frequency ω, then overtones will occur at integer multiples of ω (2ω, 3ω, etc.).*

A screenshot of a computer

Description automatically generatedA screenshot of a computer screen

Description automatically generated

**IV. Computational Results**

From our investigation, a number of interesting results were found. As detailed previously, our initial experiments aimed to examine various properties of the Gabor transform on the audio sample, Handel. Different time steps, window sizes, and window styles were explored. Overall it was shown that a moderately sized time step and window size were most desirable. This prevented undersampling and oversampling, and also avoided having poor resolution in either the time or frequency domain.

The comparison of window styles was not as straightforward, however. All windows gave fairly similar spectrogram information. This may be due to the sample data being used. In this instance, no window gave a clear advantage over another because the results were primarily exploratory. In a more rigorous analysis of some type of data, one window may be preferred due to different properties. For example, the Mexican hat window may be able to more readily detect frequencies that resemble its shape than the Shannon window.

Following these results, new audio recordings were used in an attempt to reconstruct a music score for “Mary Had a Little Lamb.” As mentioned in Figure 4, one notable difference between these recordings are the presence of different overtones. Overtones are frequencies that occur at integer multiples of a center frequency. For example, a middle A tone at 440Hz may have overtones at 880Hz and 1320Hz. In different instruments, different overtones are present at varying amplitudes. This property is known as the timbre of an instrument, and it’s why a piano and recorder playing the same note at the same frequency sound very different. This property is also impossible to view in either the time or frequency domain, meaning spectrograms are necessary to realize this difference between the recordings. When the overtones are removed, the remaining pure tone will always sound the same. In our work, we filtered out any overtones before recreating the music scores for both instruments, in order to clean up the audio data.

The reproduced music scores show that the piano and recorder were playing the same pattern of notes, but at different frequencies (this is also apparent from the spectrograms). The notes are labelled alphabetically in Figure 5, along with the frequencies they were found at. As before, frequencies were only scaled by a factor of 1/(sampling rate) to ensure that they could be translated into Hz tones.

*Figure 5. Reproduced music scores for each instrument. Note that the time axis is arbitrarily scaled and is only used to denote the sequence of notes occurring. Both the notes and frequencies for each instrument are different, though both follow the same pattern throughout the song. While the recorder score has a greater difference between each frequency, the aligned notes are each a whole step apart, as in the piano score. Alphabetical notes have been estimated based on the approximate frequencies found.*

**A screenshot of a cell phone

Description automatically generatedA screenshot of a map

Description automatically generated**

**V. Summary and Conclusions**

From the techniques used in this assignment, we have been able to explore the mechanisms of the Gabor transform, interpret spectrograms, and practically apply these principles to analyze sample data. The use of windowing/filtering, Gabor transforms, and spectrograms highlights how these techniques can be applied to a multitude of problems. Time-frequency analysis as a whole allows for the access of information that previously could not be seen. Overall, this assignment served to show how this powerful tool works, and how it can be adapted to analyze data in various scenarios.

**References**

# [1] J. Nathan Kutz, *Data-Driven Modeling & Scientific Computation: Methods for Complex*

# *Systems & Big Data.* Oxford University Press, 2013.

**Appendix A. MATLAB Functions Used**

*heaviside(x)* –  Unit step function, outputs 0 when x < 0, 1 when x > 0, and 1/2 when x = 0.

*pcolor(x,y,c)* – 2D plot of x and y values, where c corresponds to color values of each cell.

*shading interp* – Colors a cell by interpolating a color value across the cell.

*colormap map* – Applies a predefined gradient of color values to the current figure

*audioread(filename)* – Reads in an audio file and outputs a data and sampling rate

*downsample(x,n)* – Decreasing the sampling rate of x by factor of n.

*diff(x)* – Finds the difference between adjacent elements of x.

*strings(n)* – Create an empty string array of size n.

*text(x,y,str)* – Adds chosen string to x and y data points on a plot.

**Appendix B. MATLAB Code**

1 % Maxwell Weil

2 % AMATH 482

3 % HW2

4 clear; close all; clc;

5

6 % Loading sample music and sampling rate, reorienting vector

7 load handel

8 music = y';

9

10 % Setting sample number, duration of music, time vector,

11 % And unshifted/shifted frequency vectors

12 samples = length(music);

13 duration = samples/Fs;

14 time = [(1:samples)/Fs];

15 freqs = (1/duration)\*[0:(samples-1)/2 -(samples-1)/2:-1];

16 shifted\_freqs=fftshift(freqs);

17

18

19 % Defining constant for plotting static windows

20 a\_ex = 10;

21 width\_ex = 3;

22 sigma\_ex = 10;

23 tau = 4;

24

25 % Defining equations for gabor, shannon, and mexican hat windows

26 gabor\_window=exp(-a\_ex\*(time-tau).^2);

27 shannon\_window = heaviside(time-(tau-width\_ex/2))...

28 -heaviside(time-(tau+width\_ex/2));

29 mexican\_hat\_window = (1-sigma\_ex\*(time-tau).^2)...

30 .\*exp(-1\*sigma\_ex\*(time-tau).^2/2);

31

32 % Creating plots of static windows and filtered signal in

33 % Frequency and time domains

34 figure

35

36 % Gabor window subplots

37 subplot(3,3,1)

38 plot(time, gabor\_window, 'r', 'LineWidth', 2)

39 axis([0 duration -0.5 1.5])

40 title('Gabor Window', 'FontSize', 6)

41 xlabel('Time (s)', 'FontSize', 6)

42 ylabel('Amplitude', 'FontSize', 6)

43 subplot(3,3,2)

44 plot(time, gabor\_window, 'r', 'LineWidth', 2)

45 hold on

46 plot(time, gabor\_window.\*music, 'k')

47 axis([0 duration -1 1.5])

48 title('Gabor Filtered Signal', 'FontSize', 6)

49 xlabel('Time (s)', 'FontSize', 6)

50 ylabel('Amplitude', 'FontSize', 6)

51 subplot(3,3,3)

52 plot(shifted\_freqs,abs(fftshift(fft(gabor\_window.\*music)))...

53 /max(abs(fft(gabor\_window.\*music))),'m');

54 axis([-Fs/2 Fs/2 0 1])

55 title('Gabor Filtered Signal (Frequency)', 'FontSize', 6)

56 xlabel('Frequency (Hz)', 'FontSize', 6)

57 ylabel('Relative Amplitude', 'FontSize', 6)

58

59 % Shannon window subplots

60 subplot(3,3,4)

61 plot(time, shannon\_window, 'r', 'LineWidth', 2)

62 axis([0 duration -0.5 1.5])

63 title('Shannon Window', 'FontSize', 6)

64 xlabel('Time (s)', 'FontSize', 6)

65 ylabel('Amplitude', 'FontSize', 6)

66 subplot(3,3,5)

67 plot(time, shannon\_window, 'r', 'LineWidth', 2)

68 hold on

69 plot(time, shannon\_window.\*music, 'k')

70 axis([0 duration -1 1.5])

71 title('Shannon Filtered Signal', 'FontSize', 6)

72 xlabel('Time (s)', 'FontSize', 6)

73 ylabel('Amplitude', 'FontSize', 6)

74 subplot(3,3,6)

75 plot(shifted\_freqs,abs(fftshift(fft(shannon\_window.\*music)))...

76 /max(abs(fft(shannon\_window.\*music))),'m');

77 axis([-Fs/2 Fs/2 0 1])

78 title('Shannon Filtered Signal (Frequency)', 'FontSize', 6)

79 xlabel('Frequency (Hz)', 'FontSize', 6)

80 ylabel('Relative Amplitude', 'FontSize', 6)

81

82 % Mexican hat window subplots

83 subplot(3,3,7)

84 plot(time, mexican\_hat\_window, 'r', 'LineWidth', 2)

85 axis([0 duration -0.5 1.5])

86 title('Mexican Hat Window', 'FontSize', 6)

87 xlabel('Time (s)', 'FontSize', 6)

88 ylabel('Amplitude', 'FontSize', 6)

89 subplot(3,3,8)

90 plot(time, mexican\_hat\_window, 'r', 'LineWidth', 2)

91 hold on

92 plot(time, mexican\_hat\_window.\*music, 'k')

93 axis([0 duration -1 1.5])

94 title('Mexican Hat Filtered Signal', 'FontSize', 6)

95 xlabel('Time (s)', 'FontSize', 6)

96 ylabel('Amplitude', 'FontSize', 6)

97 subplot(3,3,9)

98 plot(shifted\_freqs,abs(fftshift(fft(mexican\_hat\_window.\*music)))...

99 /max(abs(fft(mexican\_hat\_window.\*music))),'m');

100 axis([-Fs/2 Fs/2 0 1])

101 title('Mexican Hat Filtered Signal (Frequency)', 'FontSize', 6)

102 xlabel('Frequency (Hz)', 'FontSize', 6)

103 ylabel('Relative Amplitude', 'FontSize', 6)

104 %% Effect of Different Window Sizes on Spectrogram

105

106 % Defining window size constants for sliding gabor window

107 a\_list = [1, 10, 100, 10000];

108

109 % Creating time steps to move window along

110 tslide=0:0.01:duration;

111

112 % Initializing spectrogram matrix

113 gab\_spec\_music = zeros(length(tslide),samples);

114

115 % Looping through different sized windows for gabor-filtered spectrogram

116 figure

117 for i = 1:length(a\_list)

118

119 % Looping through time step vector

120 for j=1:length(tslide)

121

122 % Creating gabor window at time step

123 gabor\_window=exp(-a\_list(i)\*(time-tslide(j)).^2);

124

125 % Filtering signal in time

126 gab\_filt\_music=gabor\_window.\*music;

127

128 % Transforming signal to frequency

129 freq\_gab\_filt\_music=fft(gab\_filt\_music);

130

131 % Adding frequency data to spectrogram at each time step

132 gab\_spec\_music(j,:) = fftshift(abs(freq\_gab\_filt\_music));

133

134 end

135 % Plotting spectrograms for each sized window

136 subplot(2,2,i)

137 pcolor(tslide,shifted\_freqs,gab\_spec\_music.')

138 ylim([0 max(freqs)])

139 title(['Gabor Filtered Spectrogram with a = ', num2str(a\_list(i))])

140 ylabel('Frequency (Hz)')

141 xlabel('Time (s)')

142 shading interp

143 colormap hot

144 end

145 %% Effect of Step Size on Spectrogram

146

147 % Defining window size constant

148 a = 100;

149

150 % Creating varying time steps to move window along

151 step\_list = [0.05, 0.1, 0.5, 1];

152

153 % Looping through different sized time steps for gabor-filtered spectrogram

154 figure

155 for i = 1:length(step\_list)

156

157 % Setting new time step size

158 tslide=0:step\_list(i):duration;

159

160 % Initializing spectrogram matrix

161 gab\_spec\_music = zeros(length(tslide),samples);

162

163 % Looping through time step vector

164 for j=1:length(tslide)

165

166 % Creating gabor window at time step

167 gabor\_window=exp(-a\*(time-tslide(j)).^2);

168

169 % Filtering signal in time

170 gab\_filt\_music=gabor\_window.\*music;

171

172 % Transforming signal to frequency

173 freq\_gab\_filt\_music=fft(gab\_filt\_music);

174

175 % Adding frequency data to spectrogram at each time step

176 gab\_spec\_music(j,:) = fftshift(abs(freq\_gab\_filt\_music));

177

178 end

179 % Plotting spectrograms for each sized window

180 subplot(2,2,i)

181 pcolor(tslide,shifted\_freqs,gab\_spec\_music.')

182 ylim([0 max(freqs)])

183 title({'Gabor Filtered Spectrogram'; 'with time step = '; ...

184 num2str(step\_list(i)))

185 ylabel('Frequency (Hz)')

186 xlabel('Time (s)')

187 shading interp

188 colormap hot

189 end

190 %% Effect of Different Window Types on Spectrogram

191

192 % Defining optimized constants for shannon and mexican hat windows

193 a = 100;

194 width = 0.1;

195 sigma = 100;

196

197

198 % Initializing spectrogram matrices for each window type

199 gab\_spec\_music = zeros(length(tslide),samples);

200 shan\_spec\_music = zeros(length(tslide),samples);

201 mex\_spec\_music = zeros(length(tslide),samples);

202

203 % Looping through time steps, with optimized

204 % Window sizes, for spectrogram comparison

205 figure

206 for j=1:length(tslide)

207

208 % Creating each window at specified time step

209 gabor\_window=exp(-a\*(time-tslide(j)).^2);

210 shannon\_window = heaviside(time-(tslide(j)-width/2))-heaviside(time-(tslide(j)+width/2));

211 mexican\_hat\_window = (1-sigma\*(time-tslide(j)).^2).\*exp(-1\*sigma\*(time-tslide(j)).^2/2);

212

213 % Filtering and transforming signal with gabor window

214 gab\_filt\_music=gabor\_window.\*music;

215 freq\_gab\_filt\_music=fft(gab\_filt\_music);

216

217 % Filtering and transforming signal with shannon window

218 shan\_filt\_music=shannon\_window.\*music;

219 freq\_shan\_filt\_music=fft(shan\_filt\_music);

220

221 % Filtering and transforming signal with mexican hat window

222 mex\_filt\_music=mexican\_hat\_window.\*music;

223 freq\_mex\_filt\_music=fft(mex\_filt\_music);

224

225 % Adding frequency data to each spectrogram at each time step

226 gab\_spec\_music(j,:) = fftshift(abs(freq\_gab\_filt\_music));

227 shan\_spec\_music(j,:) = fftshift(abs(freq\_shan\_filt\_music));

228 mex\_spec\_music(j,:) = fftshift(abs(freq\_shan\_filt\_music));

229 end

230

231 % Plotting spectrograms for each window type

232 % Gabor spectrogram

233 subplot(1,3,1)

234 pcolor(tslide,shifted\_freqs,gab\_spec\_music.')

235 ylim([0 max(freqs)])

236 title({'Gabor Filtered'; 'Spectrogram'})

237 ylabel('Frequency (Hz)')

238 xlabel('Time (s)')

239 shading interp

240 colormap hot

241

242 % Shannon spectrogram

243 subplot(1,3,2)

244 pcolor(tslide,shifted\_freqs,shan\_spec\_music.')

245 ylim([0 max(freqs)])

246 title({'Shannon Filtered'; 'Spectrogram'})

247 ylabel('Frequency (Hz)')

248 xlabel('Time (s)')

249 shading interp

250 colormap hot

251

252 % Mexican hat spectrogram

253 subplot(1,3,3)

254 pcolor(tslide,shifted\_freqs,mex\_spec\_music.')

255 ylim([0 max(freqs)])

256 title({'Mexican Hat Filtered'; 'Spectrogram'})

257 ylabel('Frequency (Hz)')

258 xlabel('Time (s)')

259 shading interp

260 colormap hot

261

262 %% PROBLEM 2, Analyzing A Music Score

263 clear; close all; clc;

264

265 % Reading in audio data and sampling frequency

266 [y,Fs] = audioread('music2.wav');

267

268 % Downsampling audio for fast computing, note that Fs is also adjusted

269 y = downsample(y,4);

270 Fs = Fs/4;

271

272 % Reorienting audio vector, setting sample number, duration of music,

273 % Time vector, and unshifted/shifted frequency vectors

274 song = y';

275 samples = length(song);

276 duration = samples/Fs;

277 time = [(1:samples)/Fs];

278 freqs = (1/duration)\*[0:samples/2-1 -samples/2:-1];

279 shifted\_freqs=fftshift(freqs);

280

281 % Setting selected filter constants and time step vector

282 a = 100;

283 a2 = 0.01;

284 tslide=0:0.1:duration;

285

286 % Initializing spectrogram, maximum amplitude, and tone

287 % At maximum amplitude matrices

288 spec\_song = zeros(length(tslide),samples);

289 max\_amplitude = zeros(length(tslide),1);

290 tone\_at\_max = zeros(length(tslide),1);

291

292 % Looping through time steps

293 for j=1:length(tslide)

294

295 % Creating window and filtering audio

296 gabor\_filter=exp(-a\*(time-tslide(j)).^2);

297 filt\_song=gabor\_filter.\*song;

298

299 % Transforming to frequency domain

300 filt\_freq\_song=fft(filt\_song);

301

302 % Finding the maximum amplitude, and the frequency at which it occurs

303 [high, idx] = max(abs(filt\_freq\_song));

304 tone\_at\_max(j) = freqs(idx);

305 max\_amplitude(j) = high;

306

307 % Creating filter in frequency domain around center frequency

308 overtone\_filter = exp(-a2\*(freqs-freqs(idx)).^2);

309

310 % Filtering out overtones around center frequency

311 pure\_freq\_song = overtone\_filter.\*filt\_freq\_song;

312

313 % Adding frequency data to each spectrogram at each time step

314 spec\_song(j,:) = fftshift(abs(filt\_freq\_song));

315 end

316

317 % Plotting spectrogram

318 figure

319 pcolor(tslide,shifted\_freqs,spec\_song.')

320 ylim([0 2400])

321 title('Recorder Spectrogram')

322 ylabel('Frequency (Hz)')

323 xlabel('Time (s)')

324 shading interp

325 colormap hot

326

327 % Finding where the maximum amplitudes are over a specified threshold,

328 % In order to determine true notes are being played

329 threshold\_logic = max\_amplitude>mean(max\_amplitude)/2;

330

331 % Removing repeated values, by finding difference between adjacent elements

332 high\_low\_shift = diff(threshold\_logic) < 0;

333

334 % Locating corresponding frequencies for each note

335 notes\_freq = tone\_at\_max(high\_low\_shift);

336

337 % Initializing text string for notes

338 notes\_text = strings(length(notes\_freq),1);

339

340 % Looping through frequncies and determining alphabetical note,

341 % This process has been shortened to only search for a few specific notes

342 for i = 1:length(notes\_freq)

343 if notes\_freq(i) < 1100 && notes\_freq(i) > 980

344 notes\_text(i) = 'B';

345 elseif notes\_freq(i) < 980 && notes\_freq(i) > 850

346 notes\_text(i) = 'A';

347 elseif notes\_freq(i) < 850 && notes\_freq(i) > 750

348 notes\_text(i) = 'G';

349 elseif notes\_freq(i) < 330 && notes\_freq(i) > 300

350 notes\_text(i) = 'E';

351 elseif notes\_freq(i) < 300 && notes\_freq(i) > 280

352 notes\_text(i) = 'D';

353 else

354 notes\_text(i) = 'C';

355 end

356 end

357

358 % Creating vector to plot notes

359 note\_scaling = linspace(1, 100\*length(notes\_freq), length(notes\_freq));

360

361 % Plotting frequencies and corresponding alphabetical note, keep in mind

362 % That the x-axis is arbitrary, and notes are not actually equally spaced

363 figure

364 plot(note\_scaling,notes\_freq, 'ko', 'MarkerSize', 10)

365 text(note\_scaling,notes\_freq+20, notes\_text)

366 axis([(min(note\_scaling)-100) (max(note\_scaling)+100)...

367 (min(notes\_freq)-300) (max(notes\_freq)+300)])

368 title('Reproduced Music Score for Piano')

369 xlabel('Relative Time (a.u.)')

370 ylabel('Frequency (Hz)')