OPTIMIZATION STRATEGIES FOR FFT USE IN MUSICAL AUDIO ANALYSIS

by Ken Lindsay
for CS505 Spring 2006
Southern Oregon University
Ashland, Oregon USA

TABLE OF CONTENTS

Optimization Strategies for FFT Use in Musical Audio Analysis

Abstract

In this paper we investigate the Fast Fourier Transform algorithm and consider some optimizations intended to make the FFT more efficient for use in analyzing digital audio signals. Two optimization strategies are considered: 1) use of alternative basis functions for the Fourier series, and 2) identifying portions of the standard FFT computation mechanism which may be unnecessary for adequate spectral analysis of the audio signal in question.

Introduction

We have used the FFT extensively in analyzing music for feature recognition. Our primary purpose is the extraction of rhythm and melody features. We have concluded that the basic nature of Fourier Analysis is sub-optimal for this work: information available at high frequencies is excessive and over specifies this part of the spectrum, and not enough frequency resolution is available at low frequencies to adequately serve our information analysis needs. Partly, this problem can be considered as a computational efficiency issue and partly it can be considered as a basic design flaw in standard Fourier analysis.

The human ear analyzes spectral information of a sound using a set of frequencies that are nonlinearly spaced in the frequency spectrum. The FFT and Fourier analysis represent a sound by using a set of frequencies that are linearly spaced. Although the two forms are mathematically equivalent, the difference between them may be quite substantial from a practical point of view.

In standard Fourier analysis, a short segment of a time domain signal is converted to the frequency domain, yielding the *spectrum* of the audio signal as a set of related sine and cosine waves. The set of output frequency components (the sine and cosine basis functions) are linearly spaced in the frequency domain. A typical example is

$$f(x) = a_0 + a_1*\cos(x) + b_1*\sin(x) + a_2*\cos(2x) + b_2*\sin(2x) + a_3*\cos(3x) + b_3*\sin(3x) \ldots \ ,$$

where the `a` and `b` coefficients are usually real numbers but can also be complex numbers. These coefficients are the *amplitude* of each component waveform. The integers inside the

parentheses are the *frequency* of each particular component waveform. The a_0 term represents any DC offset in the original signal. Adding together the set of simple waveforms recreates the original audio signal waveform to an arbitrary degree of accuracy, depending on how many sines and cosines of different frequencies are used in the Fourier series. The restriction of frequencies to integers means that all component sines and cosines are *harmonically* related. Each component fits cleanly into the original short segment from the time domain, with no spillage outside the original bounds. Figure 1 illustrates a set of sine waves and the composite waveform which is generated when the components are added together. The individual waveforms are stacked for easy viewing, but this is a graphic convention and does not represent any real feature of the waveforms *per se.* Figure 2 shows a close-up of the composite which is scaled to reveal the shape of the waveform more clearly. Notice in particular that the starting and ending points (*y* location, i.e. amplitude) of each waveform are equal, in this case equal to zero since these are sine waves. Cosine waves start and end at $y = 1.0$. The amplitude of each waveform is normalized to [-1.0, 1.0]

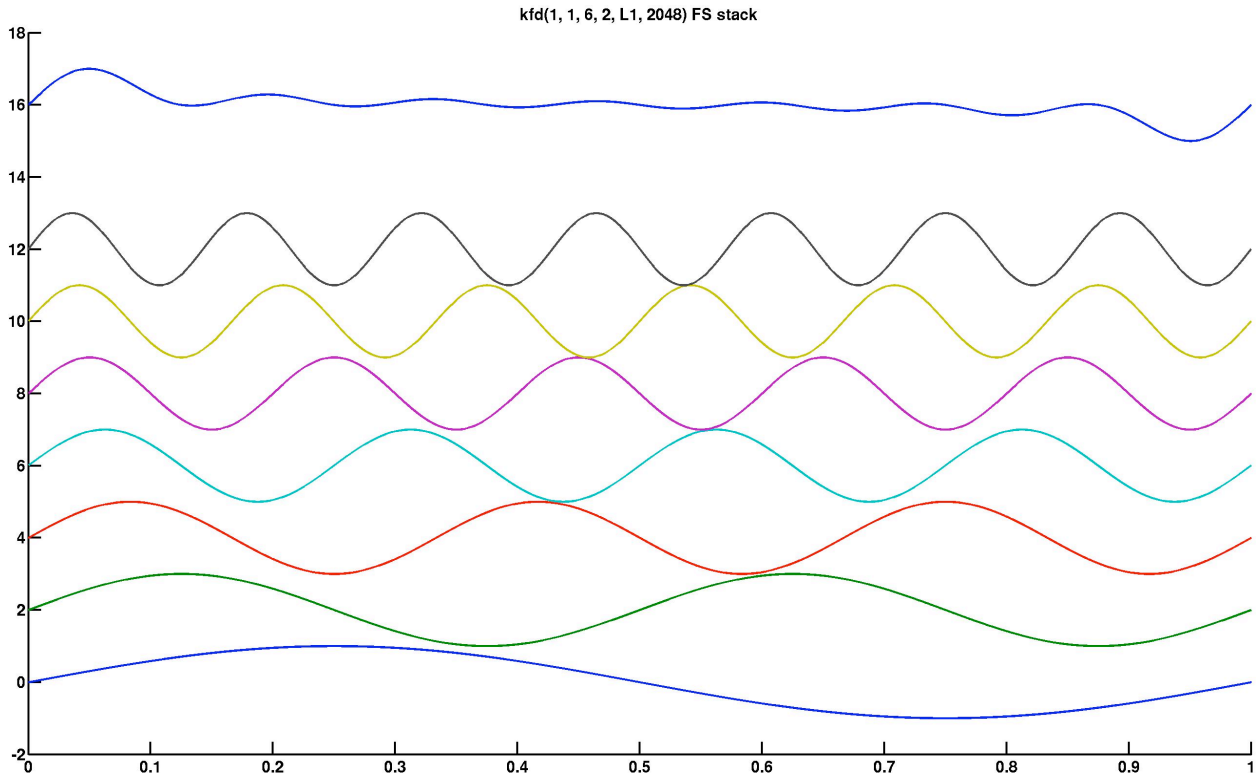


Figure 1 Seven Fourier Sine Waveforms, and the Composite Waveform

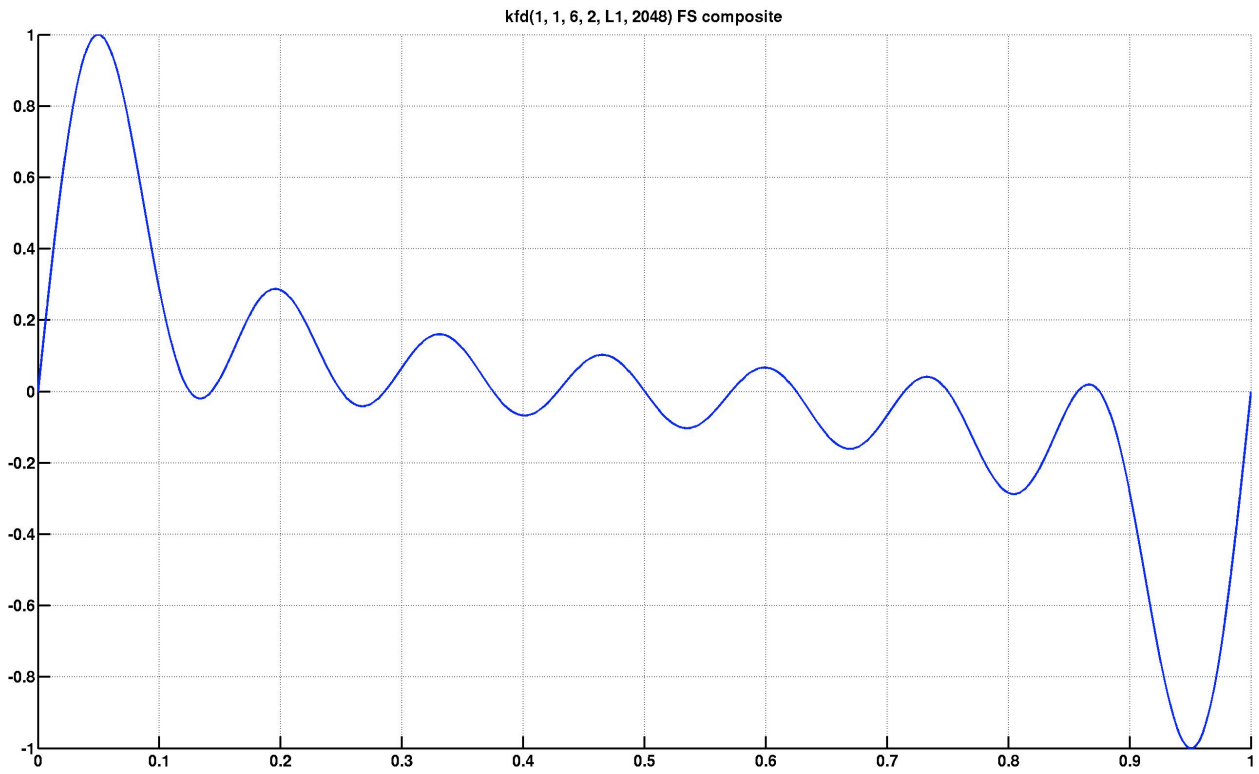


Figure 2 Close-up of Composite Waveform from Figure 1

In nonharmonic Fourier series, the basis functions are allowed to have non integer coefficients for determining the frequency of each sine or cosine wave. In this case the integer factors `2, 3, ...` inside the sine and cosine functions are free to take on other real (or complex) values rather than just integers. This has the potential to create algorithms which have exponentially spaced frequencies in the output rather than linearly spaced frequencies, mimicking the human ear. Figures 3 and 4 show an example of nonharmonic Fourier series which is analogous to Figures 1 and 2. Note that the beginning *y* location of each waveform is zero, since they are sine waves. The ending point of each waveform is not `y = 0` since the frequency (and wavelength) of the components is not restricted to integer relationship with the length of the original time domain interval. The actual formula used to generate the frequencies is

$$f_n = f_o + delta * (n - 1)^{pow} . \quad n = 1, 2, 3, \ldots . \quad pow = 1.1 . \quad delta = f_n / 10.0 .$$

The amplitude coefficients for nonharmonic Fourier series are analogous to standard Fourier series, i.e. generally they are real numbers, but they can also be complex numbers.

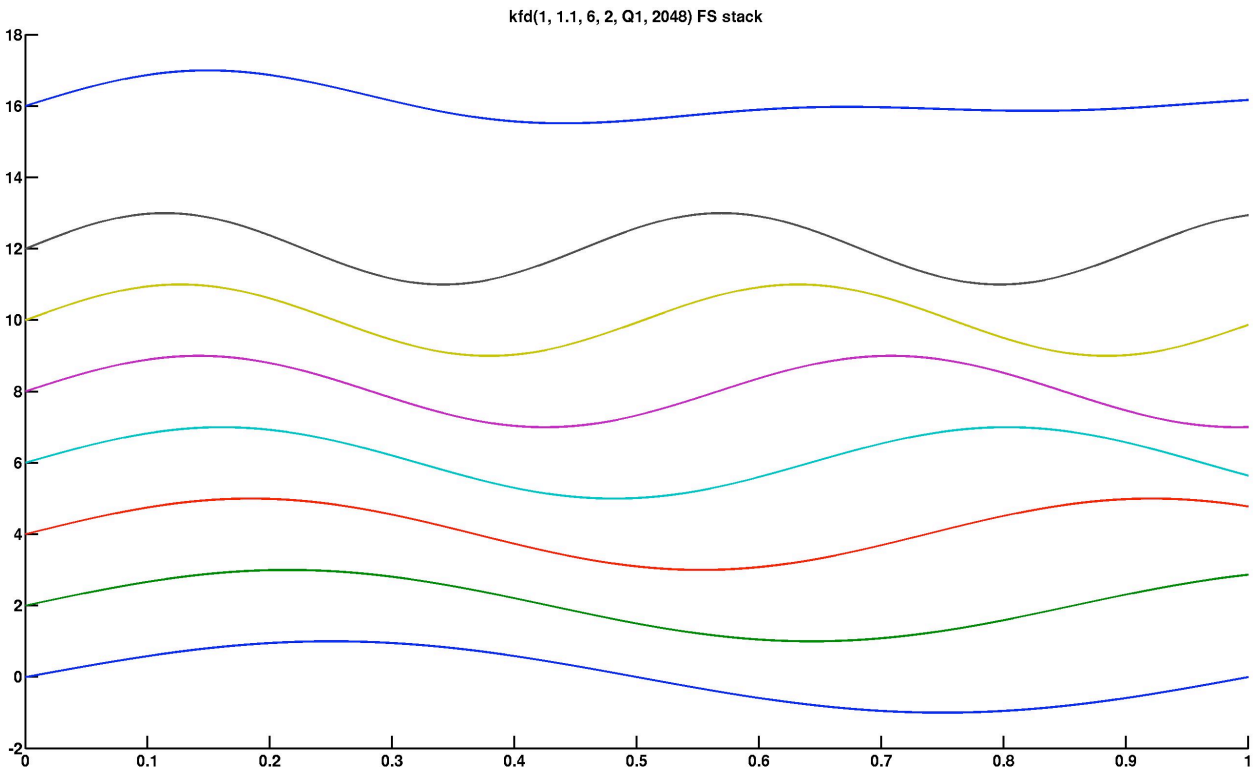


Figure 3 Nonharmonically Related Sine Waves and the Composite Waveform

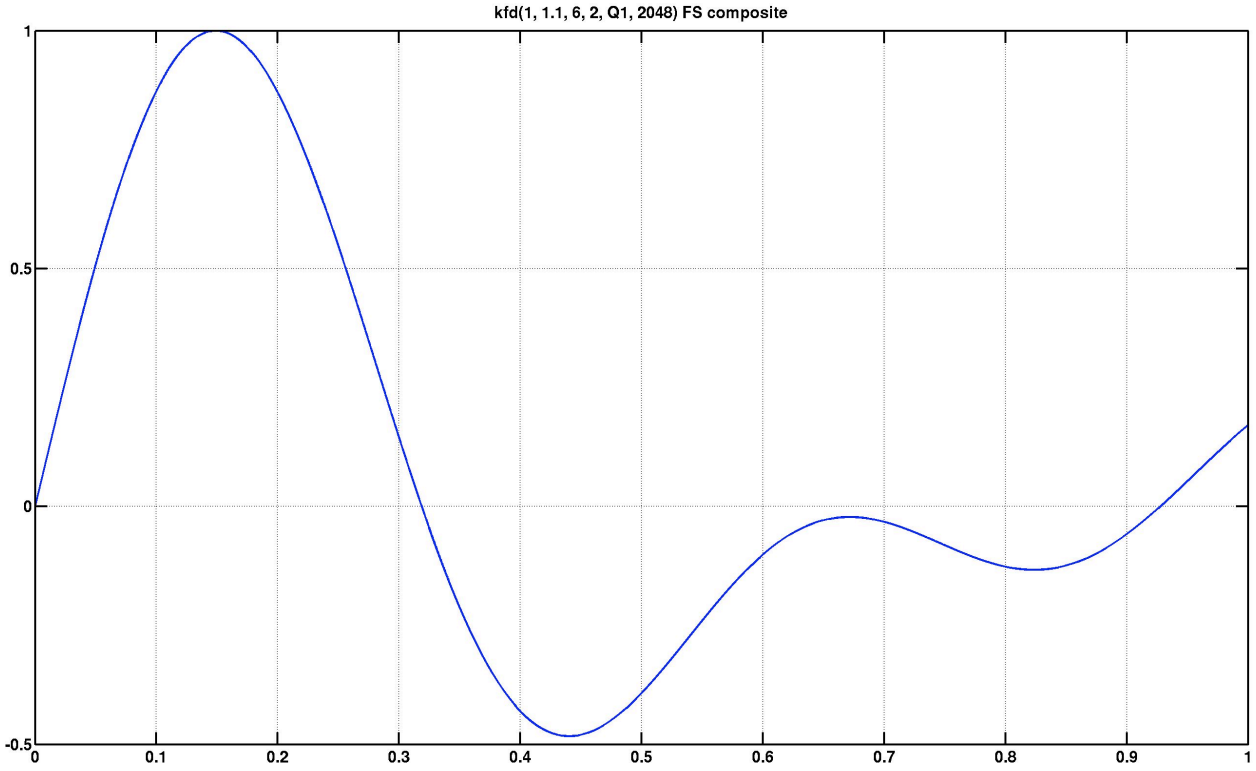


Figure 4 Close-up of Composite Waveform from Figure 3

The specific frequencies generated for the two examples are:

harmonic series: $f_n = \{ 1, 2, 3, 4, 5, 6, 7 \}$.

nonharmonic series: $f_n = \{ 1.0, 1.167, 1.357, 1.558, 1.766, 1.979, 2.196 \}$.

Note that it would be difficult for the standard Fourier approach to generate the waveform in Figure 4. It is possible, but would require many more than 7 basis waveforms, and since the end point is not equal to zero, would require cosine waves as well as sine waves. The composite waveform in Figure 2 could be generated by a nonharmonic Fourier series. Picking the frequency coefficients carefully would allow both end points to be matched more easily than using standard Fourier series to match Figure 4. Probably it would need fewer nonharmonic components (properly chosen) to generate Figure 2 than would be required of harmonic components to generate Figure 4, but we have not done this analysis yet.

Optimization Strategies

In our analysis of the first optimization strategy, we consider *nonharmonic* Fourier series, which are close mathematical cousins to *wavelets.* When properly implemented, this approach yields results which are as *correct* as the canonical Fourier series that uses harmonically related sine and cosine functions. By correct we mean that each of the two methods provides an accurate transformation of audio signals from the time domain into the frequency domain, within the confines of the mathematical assumptions of each method.

In the analysis of the second strategy, we explore the idea that some sections of the information space may be over represented by the full set of Fourier sines and cosines. For example, the FFT returns 75% of its frequency components (basis functions) in the higher frequencies (5000 Hz to 20,000 Hz). However, most of the useful *information* in the audio signal is contained in the frequencies below 5000 Hz, and indeed below about 2000 Hz. The computation of some of the higher frequencies might be omitted in the algorithm, while maintaining a sufficiently accurate spectral representation of the audio sample to be useful for information analysis and pattern recognition. Omitting these computations may reduce the "correctness" of the spectrum in the context of the Shannon-Nyquist sampling theorem and other standard DSP measures. Since we are not intending to reconstruct the signal, and are more concerned with accurate pattern recognition than with mathematical

exactitude, we think it is acceptable to ignore some of the information available, and to consider it redundant for our purposes so long as our pattern recognition needs are met.

The math behind Fourier analysis and the FFT explicitly assumes that the segment of the audio signal being processed by the algorithm is infinitely repeated throughout all time in an exact copy/paste style. This of course is never true in real life, and is one of several sources of information distortion in standard DSP processing. The length of the FFT window (number of audio samples) represents the "fundamental" frequency of the small section of audio being processed, and the frequencies of all the sines and cosines in the Fourier series for the audio waveform are integer multiples of this fundamental frequency. For example, at the CD sampling rate of 44,100 samples per second, a 2048 point FFT gives a fundamental frequency of approximately 46 Hz: 2048/44,100 ~= 46.44 . Other frequencies outside the canonical set are not represented, except approximately. One interpretation of non integer factors inside the sine and cosine functions of the Fourier series is that it is no longer true that all the basis functions are harmonic multiples of the FFT window "fundamental". Basis functions are merely the simple sine and cosine component waveforms which, when added together, give the original audio waveform.

Non Harmonic Fourier Series

Fourier series analysis in computer music is a method of representing complicated audio waveforms in terms of simpler waveforms, specifically sine and cosine functions. The original audio waveforms are typically composed of data points represented as 16 bit numbers (or equivalent) that are equally spaced in time. These digitized samples are the measured voltage or power of the original analog audio waveform at each point in time. A time series plot is simply the graph of the set of samples on the *y* axis, one for each location on the *x* axis, which is the elapsed time in the musical waveform. Connecting the dots gives a clear view of the shape of the audio waveform. Figures 1 through 4 are synthetic examples which are also typical of the overall concept of time series plots. We next consider an interpretation of these waveforms as points in an N-dimensional space, where N is the FFT size.

Function Space

We assume the reader is familiar with the standard 3D representation of geometry in terms of the *x, y,* and *z* axes, and representation of points in the familiar world in terms of (x, y, z) components. The analysis of audio waveforms (and signals in general) can be handled similarly to the usual representation of 3D geometry with its vertices, polygons and locations in 3-space. Whereas a single point in 3-space needs 3 numbers, an *x, y,* and *z* value, to specify its location, an audio waveform can be represented as a set of N elements *n1, n2, n3, ... nN* that are the terms of the Fourier series used to approximate the shape of the waveform. If *N* is large enough, any waveform (with a few sensible restrictions) can be represented completely accurately by its Fourier series. It is a correct view to consider the Fourier amplitude coefficients for the sine and cosine waves { a_1, a_2, a_3, ... a_N , b_1, b_2, b_3 ... b_N } as (x, y, z, ...) coordinates of the waveform in an N-dimensional space where each point in the space is a different waveform. Thus, if you have looked at FFT spectral plots and understood them as a list of frequencies in the spectrum of the sound, you have been doing analysis in a 1024 dimensional space (or however large your FFT window is). Although the mathematical foundations of function spaces are deep and subtle (and for most people, difficult), understanding them in more familiar terms of audio waveforms is relatively intuitive and simple.

A convenient detail of the standard 3D (x, y, z) coordinate system is that *x, y,* and *z* are mutually perpendicular, and the basis vectors { *x* = (1, 0, 0), *y* = (0, 1, 0), *z* = (0, 0, 1) } are one unit long by the measure of 3D space. This convention simplifies the mathematical processing of 3D data. Similarly, sines and cosines of a Fourier Series are orthonormal: mutually perpendicular and one unit long (amplitude = one), by the measures of the N-dimensional function space of the Fourier transform. A benefit of orthonormality is that waveforms (function objects, or points in the Fourier transform function space) with similar shapes also have similar sets of Fourier coefficients { a_1, a_2, a_3, ... a_N , b_1, b_2, b_3 ... b_N } that describe their "location" in the function space. For pattern recognition purposes, this simplifies the task of classifying the type of waveform that is being analyzed. Just as it can determined by simple subtraction whether a set of points in 3D space are near the point (x, y, z) = (10, 0, 0) (i.e., near the location on the *x* axis at x = 10), so can an audio

waveform be classified as being "near" a point in N-dimensional Fourier series space by comparing its coordinates (Fourier coefficients) to the known location. The known location may be determined by a note event with a fairly unique set of Fourier coefficients, such as a trumpet playing a B flat note above middle C, which would be a different location in the function space than would be a piano playing the same pitch. The location of a waveform in Fourier series space is largely determined by its timbre and fundamental frequency. The loudness (amplitude) of the musical waveform would also play a role, but this can been seen as analogous to changing the size of a 3D geometrical object without changing its shape. Such differences can be ignored for some purposes, such as distinguishing the piano from the trumpet. These differences would *not* be ignored for more sophisticated pattern recognition, such as determining the "mood" of a section of music by its loudness and whether it is punchy and bright vs smooth and mellow.

(Young, 2001) presents an analysis of the mathematics behind these function space ideas. The various theorems are concerned with mathematical properties such as convergence of one form (e.g. Fourier series) to another (the original audio waveform) and other details of the mathematical relationships amongst such objects. It is not at all clear from Young's treatment exactly how one might pursue the strategy to make an algorithm to do this alternative nonharmonic FFT. He is mostly concerned with proof of the mathematical correctness of the approach. This is, of course, proper behavior for a mathematician, but can be frustrating for applied information scientists and engineers who want some practical and easy to use technique to apply in their research or production work.

In the creation of the original Cooley-Tukey FFT algorithm, (Brigham, 1974) notes that the mathematical approach behind the FFT was known and had been published in several forms in the mathematical literature prior to the development of the Cooley-Tukey algorithm. The main reason that Cooley-Tukey has become popular and well known is that Richard L. Garwin, who was a high level researcher serving on the President's Scientific Advisory Committee, pushed Cooley and Tukey to make a practical product rather than just another research paper. (Press, 2002) notes that the math behind algorithm can actually be traced back to 1805 and the work of the great mathematician Gauss. What's needed to turn nonharmonic Fourier analysis into a practical tool is a powerful manager to persuade these

mathematicians into producing an algorithm analogous to the Cooley-Tukey (or Danielson-Lanczos, or Runge-Konig, or Gauss) algorithm. There are numerous papers related to non-harmonic Fourier series published in the areas of control systems and optimization. Perhaps there is a jewel waiting to be discovered in the research literature, or developed from such work.

Reducing Computation in the Standard FFT Algorithm

This section will look at tiling in the time/frequency space, and consider how the higher frequency ranges in the FFT really represent small FFTs being repeated very often in time. Each of these sub-FFTs looks at the high frequency content of the signal. Since the spectrum of the signal changes slowly relatively to the high frequency sines and cosines of the FFT, this could be optimized merely by omitting say 3 out of 4 (or 9 out of 10, or ...) of these sub-FFTs, and assuming that the missing time/frequency tiles are equal to the ones that are computed. Heuristic determination of the accuracy of the results can be verified as needed by comparing the sub-FFTs and adapting the inner loop of the algorithm based on the current spectrum of the signal. This could happen at numerous frequency ranges.

One potential downfall of this idea is that it may be true that the progressive stages of the FFT algorithm keep the frequency information from one part of the spectrum inextricably tangled with frequencies that are fairly distant in the spectrum from the frequencies we want to ignore. Hence, in order to get the desired frequency information, it may be mandatory to explicitly compute all frequency information. Investigating this concept is a fairly large and technically challenging project that is beyond the current scope.

Another possibility is that there is redundancy shared between FFTs which are tiled together to produce a Short Time Fourier Transform (STFT). The analysis of factoring of the transform matrix could be done, and it might not be too difficult. More problematic is that each STFT slice is first windowed by a gaussian function to reduce aliasing artifacts which are introduced by the beginning and end of the time segment which is undergoing Fourier transform. Figure 5 shows an example of an audio waveform of a small segment of time, a gaussian window function of the same length, and the composite result which is then

processed by the FFT algorithm. Note that the final waveform tapers off to zero at the beginning and end of the interval, but still retains very similar shape to the original audio.

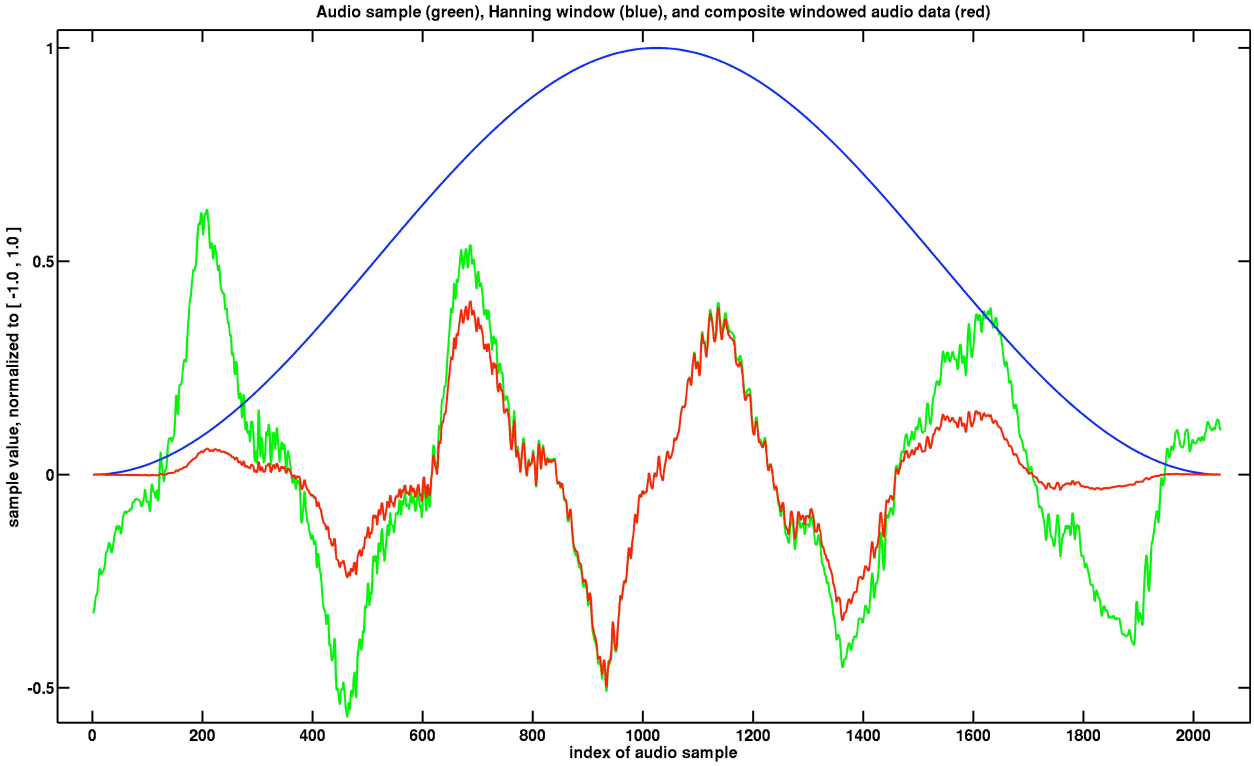


Figure 5 Audio Data, Window Function and Composite Result for FFT

Conclusions

We have considered alternatives to the standard FFT view of reality when working with digital audio samples. We are most interested in musical audio analysis and pattern recognition. Unlike speech recognition which can have good results even when ignoring frequencies above 2000 Hz, music analysis generally should not be limited in the frequency domain due to the presence of important information in the higher frequencies. The expanded frequency band for music inherently uses 4 to 10 times as many resources in terms of computational cost as an adequate speech recognition system might require. Not all of the high frequency information computed by the FFT algorithm or Fourier Analysis in general is necessary for purposes of computational pattern recognition.

An ideal solution would be to design a method similar to Fourier analysis which uses fewer frequency component basis functions (or "bins" as they are often called) in the higher

frequency range of the spectrum, and more closely spaced basis functions in the lower frequency range. It might be optimal to mimic the human ear and use basis functions which are spaced exponentially, but there may be other methods of generating the deltas between frequency bins that are equally good for the purpose of computer pattern recognition and information extraction. It is quite likely that a matrix factoring scheme could be designed which is analogous to the Cooley-Tukey/Danielson-Lanczos/Gauss approach.

The proper way to look at the sine and cosine basis functions of Fourier analysis is that they are part of a more coherent framework, *complex exponential functions,* which are factored in the sub-matrices of the FFT into simpler, more efficient forms. The factoring of the the complex exponentials is technically fairly advanced, but in essence is not much different from factoring integers into their components: e.g. 4 = 2 x 2 , 12 = 3 x 2 x 2 and so on. (Young, 2001) shows that there are infinitely many ways to factor such a decomposition of function space, and this is with the mathematical restriction of true convergence, correctness and so on. It is likely that a nonharmonic factoring scheme could be designed which is useful for musical audio analysis and also maintains mathematical correctness. It is virtually certain that such a scheme (or infinitely many schemes) could be found by relaxing the mathematical strictures of the system, but maintaining sufficient accuracy for most practical purposes..

Appendix 1: Explanation of Figures

The Matlab script kfd.m produces a set of related sine waves, plots the individual waves together with the summed composite waveform in one graph, a specgram of the FFT of the summed waveform in another graph, and a third graph of the composite waveform. Two strategies are implemented for determining the frequency relationship amongst the component sine waves. A linear set with constant delta frequency can be constructed, or a set of waves which are related by a quadratic change in the delta frequency. Two methods of specifying the linear set are available: 1) specify two frequencies, and a number of sine waves for the algorithm to fit in the frequency band, and 2) specify a base frequency, a specific delta frequency, and the number of sine waves. The quadratic method is specified by a base frequency, the number of frequencies to generate, and a real number coefficient, ***pow***, to use for an exponent in the formula: $f_n = f_0 + delta * (n - 1)^{pow} \quad n = 1, 2, 3, \ldots$. The script prints a vector of the frequencies generated for each run of the algorithm.

There are several sources of resolution errors and distortion of the information as generated and analyzed by this simple DSP method. A time vector ***t*** is created with 44,100 slots, and normalized to one second, yielding a time granularity of about 23 microseconds. The f_n value is used as a multiplier for the ***t*** vector, re-normalizing the ***t*** vector from a one second period (1 Hz) to the period determined by f_n . Except at low frequencies, this process typically yields a waveform which is only marginally sinusoidal -- sharp corners in the waveforms are visible artifacts in the time series plots. This is a form of sampling error and is related to the granularity of time in the system. While the Shannon/Nyquist sampling theorem "proves" that having 2 samples per cycle of the waveform for a particular frequency, we find that this coarse sampling rate produces many artifacts. (Hamming, 1983) warns of this type of error, and much of the engineering work in designing such a DSP system is spent on identifying and reducing these artifacts. The fact that professional recording studios generally use 24 bit samples (rather than 16 bit as a CD uses), and sampling rates up to 192 KHz (rather than 44.1 KHz CD rate, or 48 KHz DAT format sampling rate) clearly indicates that real people can tell the difference. Shannon and Nyquist were primarily concerned with intelligibility for telephone speech, and minimizing costs in such a system. This is not appropriate for a high quality musical audio system.

Figures A1, A2 and A3 show plots output by the kfd code. The first two are time series plots of generated Fourier frequency sets. The third shows the distribution of frequencies for a quadratic Fourier set. The analogous plot for a linear frequency set would be a straight line, and we omit that figure.

Figures A4, A5 and A6 show Fourier spectral plots of the same set of frequencies, processed by using FFTs of different lengths. The actual frequencies (in Hz) are

{ 110, 117.3, 133.8, 157.5, 187.4, 223.1, 264.2, 310.4, 361.5, 417.3, 477.5, 542.2, 611.1, 684.1, 761.2, 842.2 }

The frequencies are linearly spaced, within the roundoff error. Note how longer FFTs give better resolution of the spectral lines, and fewer artifacts. In an ideal world, the spectral plot would show 16 perfectly sharp red lines, evenly spaced in a field of pure blue. In the real world of DSP, we encounter spillage of energy from one part of the spectrum to other parts. These are represented by the green/yellow "sky", the yellow spaces between red spectral lines, and the red line at the very bottom which represents extremely low frequencies not present in the original signal. All of these are artifacts of sampling resolution and FFT window length. In a real audio signal we would also encounter artifacts related to the fact that the segment being processed does *not* extend infinitely in both directions for all time, as Fourier Analysis and the FFT explicitly assume.

Figures A7 through A12 show specgrams for 50 linearly spaced frequencies between 100 Hz and 2500 Hz. The effect of better resolution with longer FFTs is again clearly visible. Additionally Figures A7 and A8 show examples of severe distortion due to interaction between FFT window size and the sampled waveform data. We have not analyzed this effect thoroughly at this time. There are also FFT artifacts visible in Figures A8 through A12, although they are far less severe

Figures A13 through A20 show a similar set of specgrams for a quadratically spaced frequency set. The exponent for the frequency difference formula is 1.5 for this set. The FFT artifacts in the 1Ks and 2Ks specgrams are severe, but quite different from the artifacts in the linearly spaced frequency set. As the FFT length increases, notice how the lower frequencies are progressively better separated. Notice also that the frequency lines are not uniform: some them are sharper and better defined than others with distinct red centers and

less of the yellow “fringe” above and below. We have not analyzed this effect yet either, but believe that it is caused by interaction between the frequency of the component waveform and the length of the FFT window.

Fourier Series and FFT Results

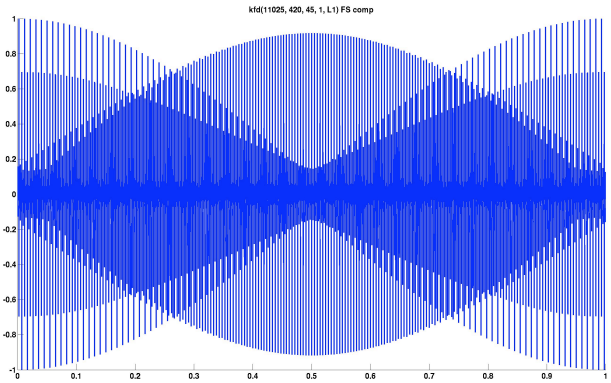


Figure A1. Fourier Series Composite Waveform

with Low Frequency Amplitude Artifacts

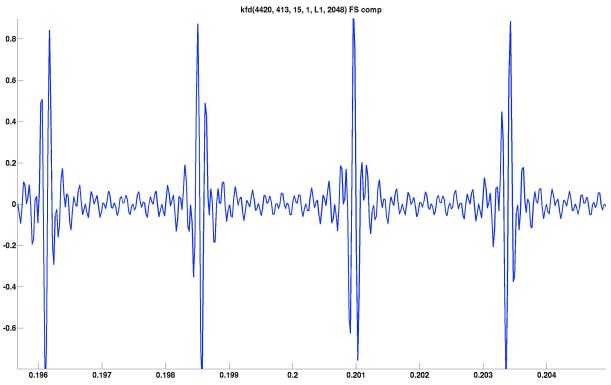


Figure A2. Close-up of Composite Waveform

with Low Frequency Waveform Artifacts

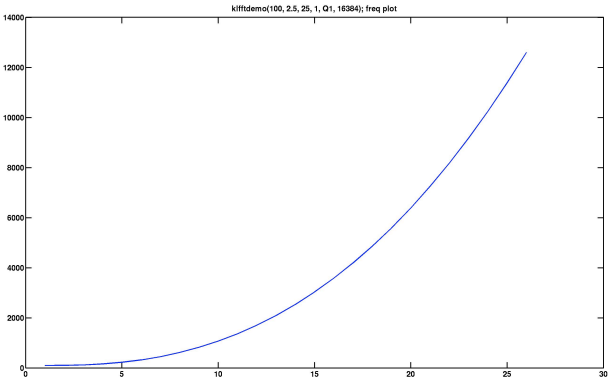


Figure A3. Frequency Distribution Curve

*Quadratic: delta * (n -1)^2.5*

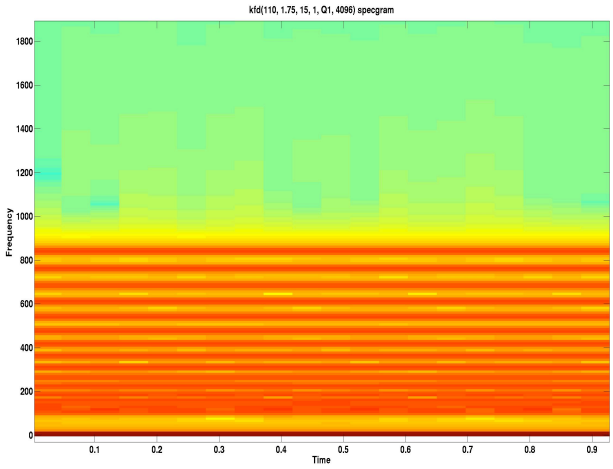


Figure A4 4096 Point FFT Spectrum

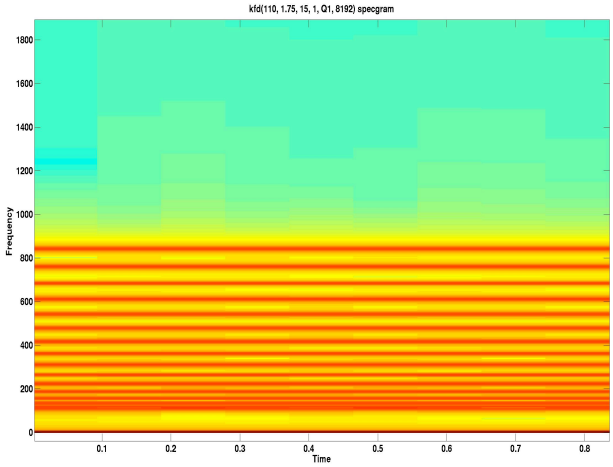


Figure A5 8192 Point FFT Spectrum

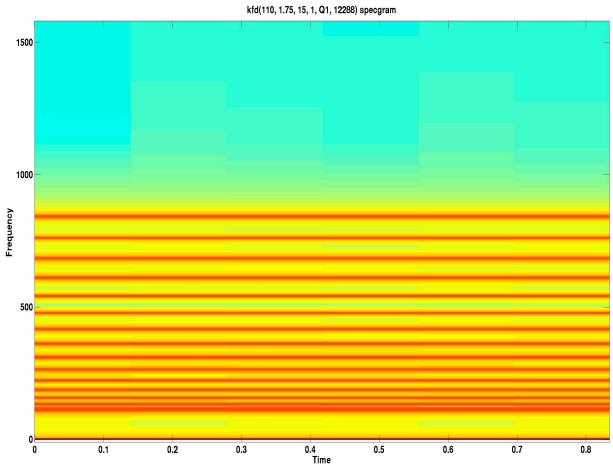


Figure A6 12288 Point FFT Spectrum

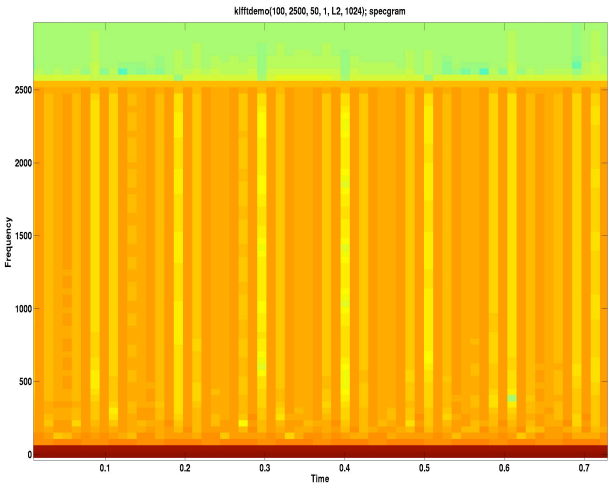


Figure A7 1024 Point FFT Spectrum

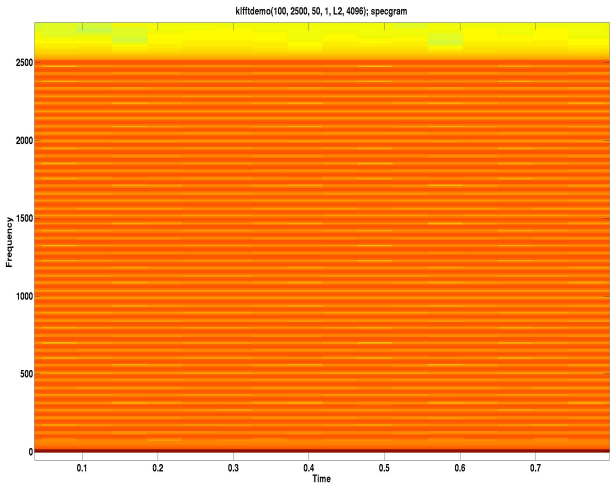


Figure A10 4096 Point FFT Spectrum

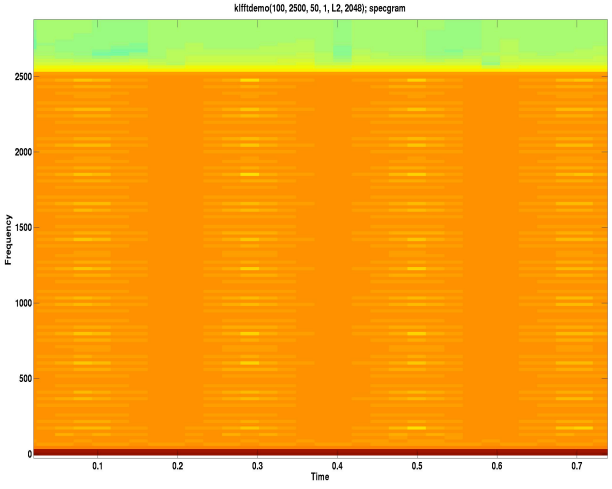


Figure A8 2048 Point FFT Spectrum

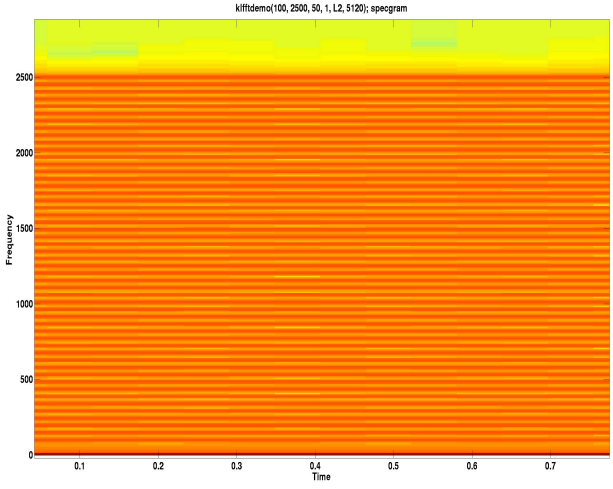


Figure A11 5120 Point FFT Spectrum

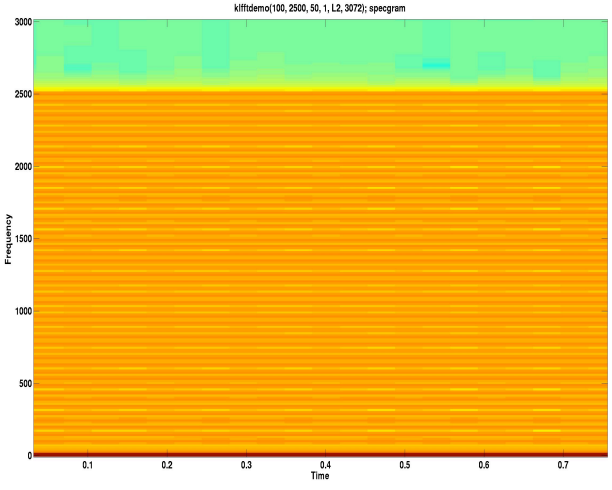


Figure A9 3072 Point FFT Spectrum

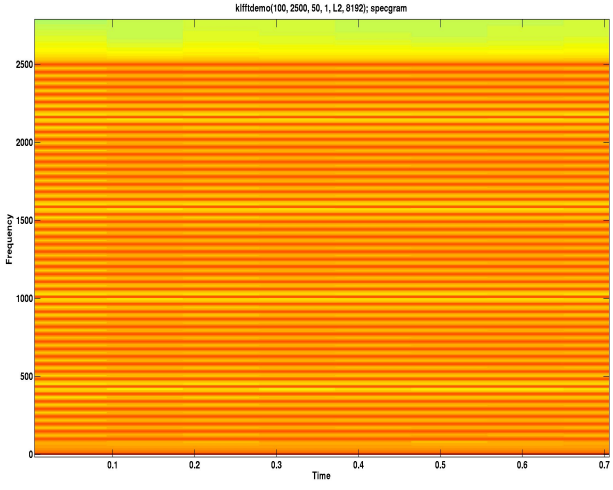


Figure A12 8192 Point FFT Spectrum

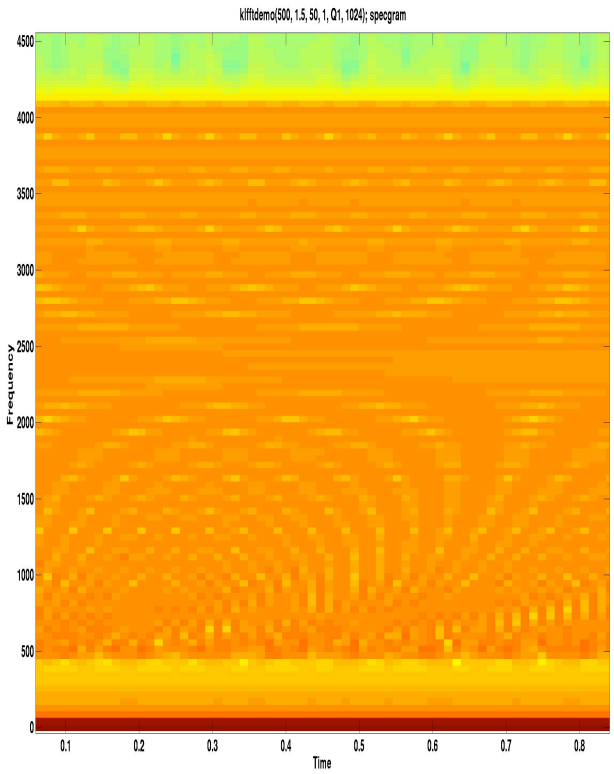


Figure A13 1024 Point FFT Spectrum

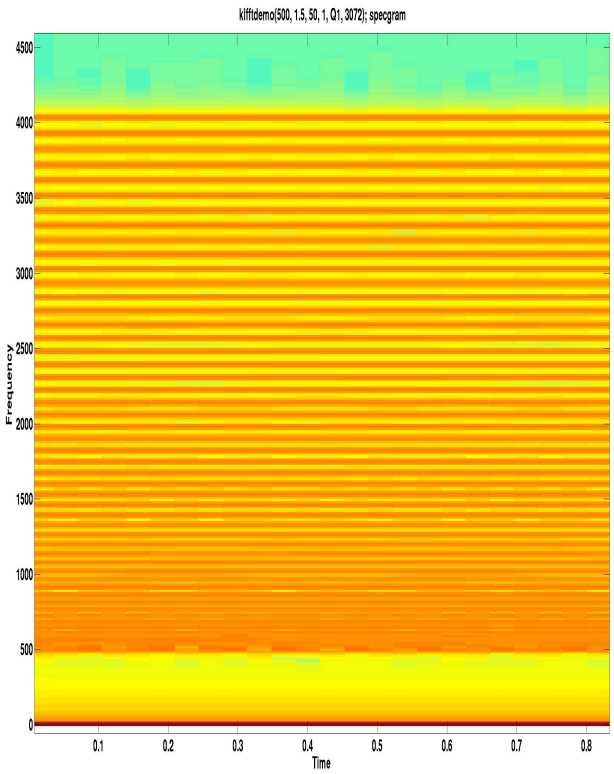


Figure A15 3072 Point FFT Spectrum

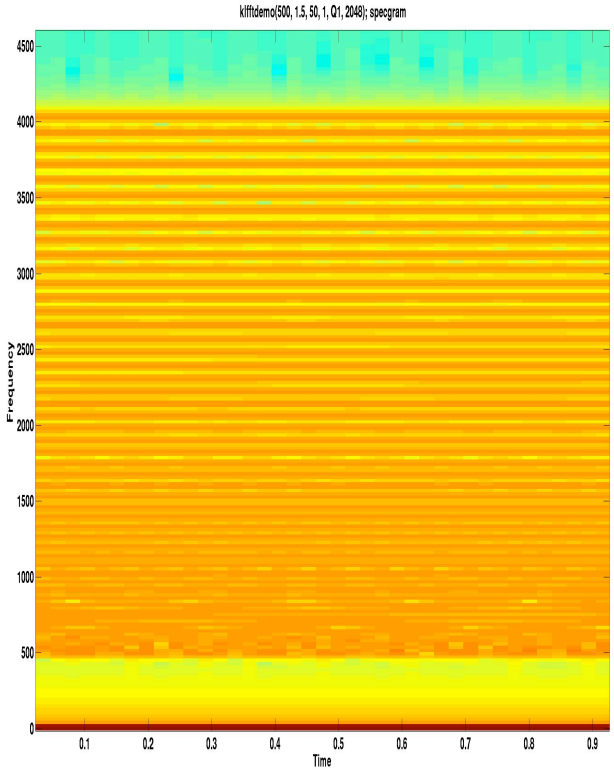


Figure A14 2048 Point FFT Spectrum

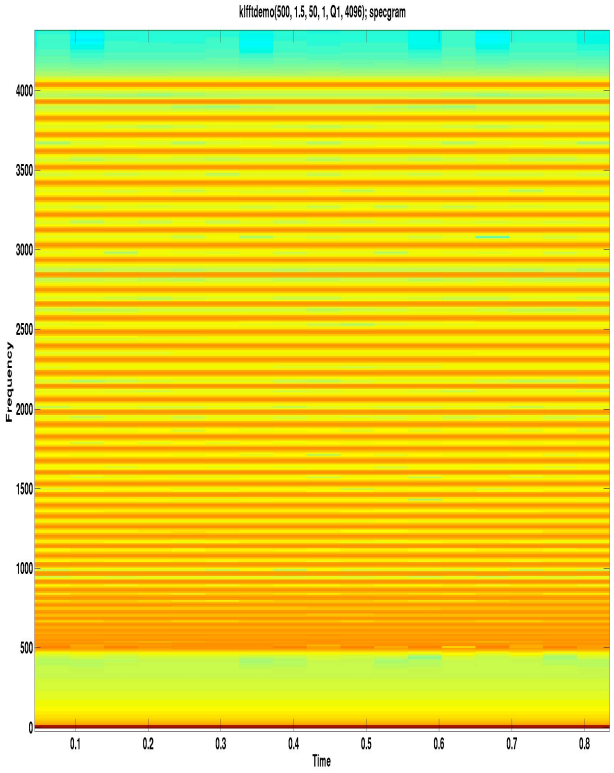
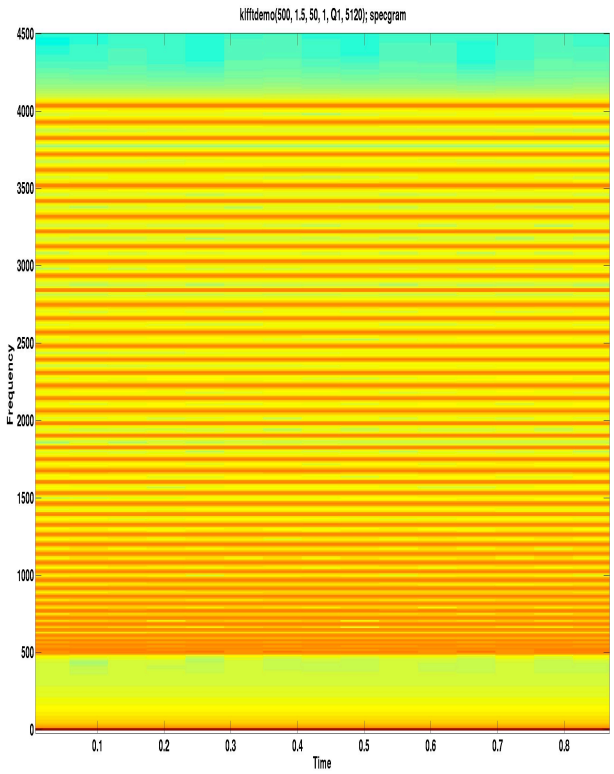


Figure A16 4096 Point FFT Spectrum



FigureA17 5120 Point FFT Spectrum

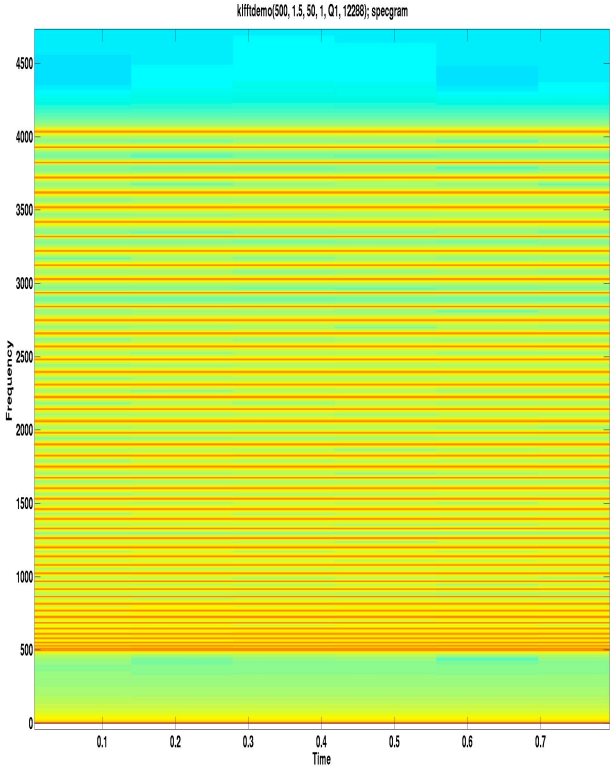


Figure A19 12288 Point FFT Spectrum

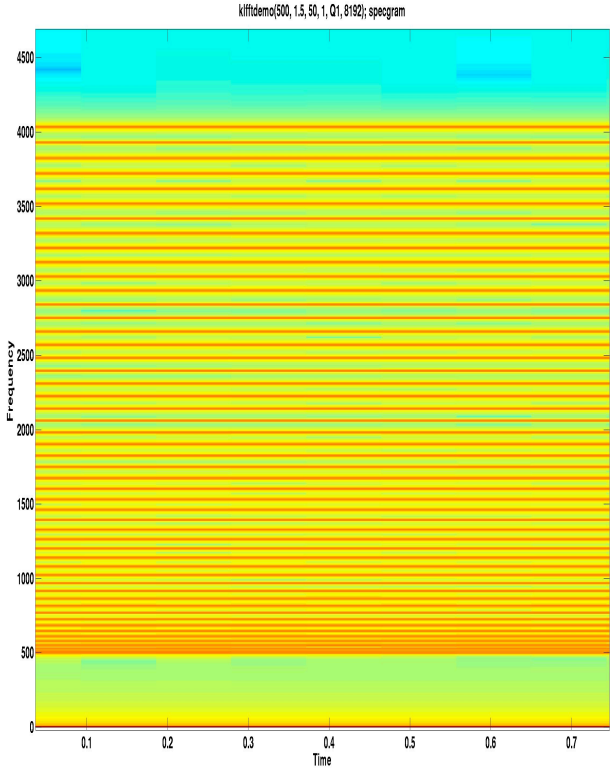


Figure A18 8192 Point FFT Spectrum

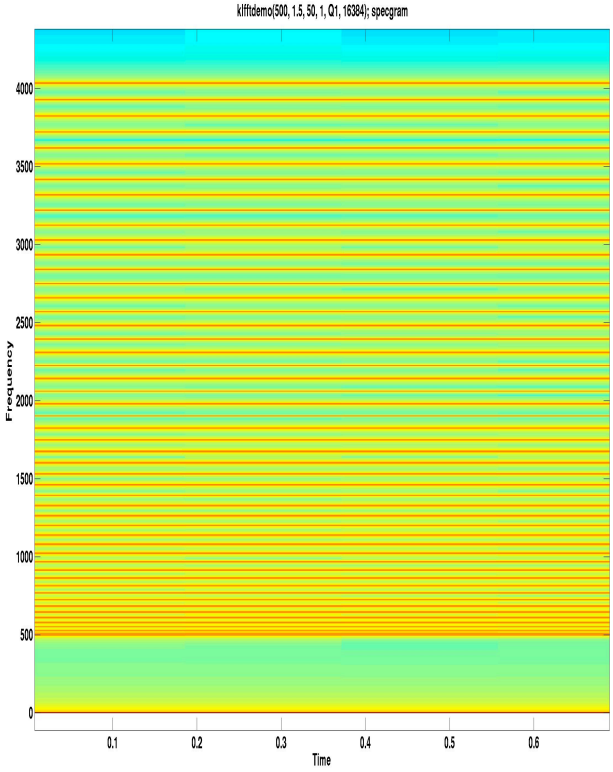


Figure A20 16384 Point FFT Spectrum

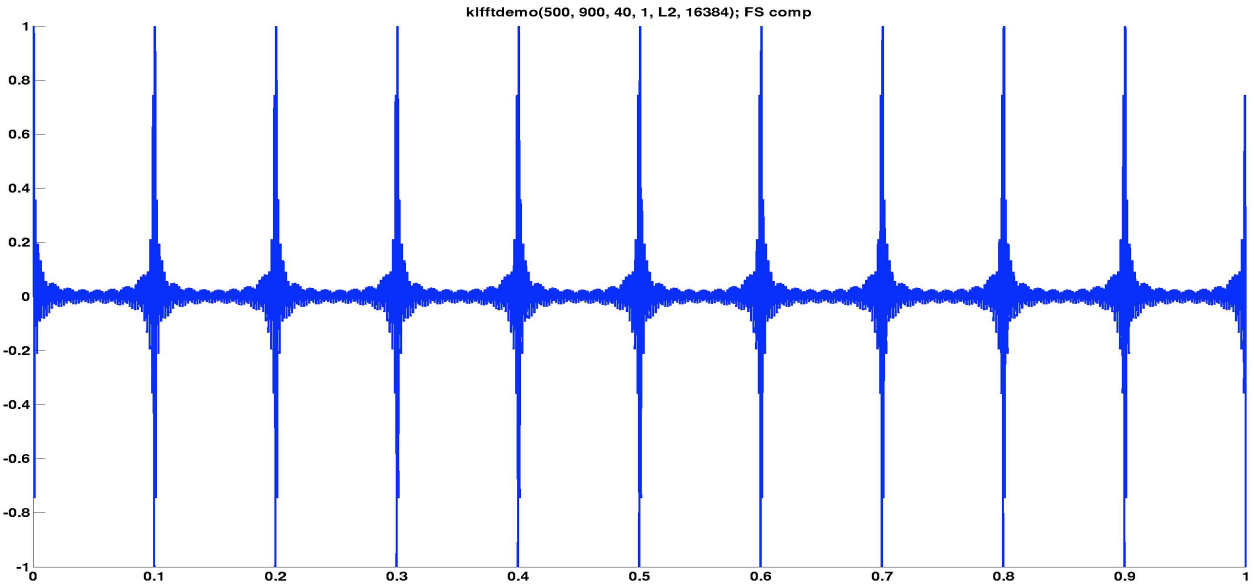


Figure A21 Composite waveform Showing Low Frequency Artifacts

This waveform is the composite of 40 frequencies spaced equally between 500 Hz and 900 Hz. The 0.1 second spikes (10 Hz) are clear examples of low frequency artifacts. Additionally, there is a kind of ripple in the low amplitude sections between the spikes. This ripple has a frequency about 8 times the frequency of the spikes, making it an 80 Hz (approximately) artifact. We have not analyzed this in detail but it is probably caused by sampling granularity in the component waveforms. Figure A22 shows a close-up of the ripple.

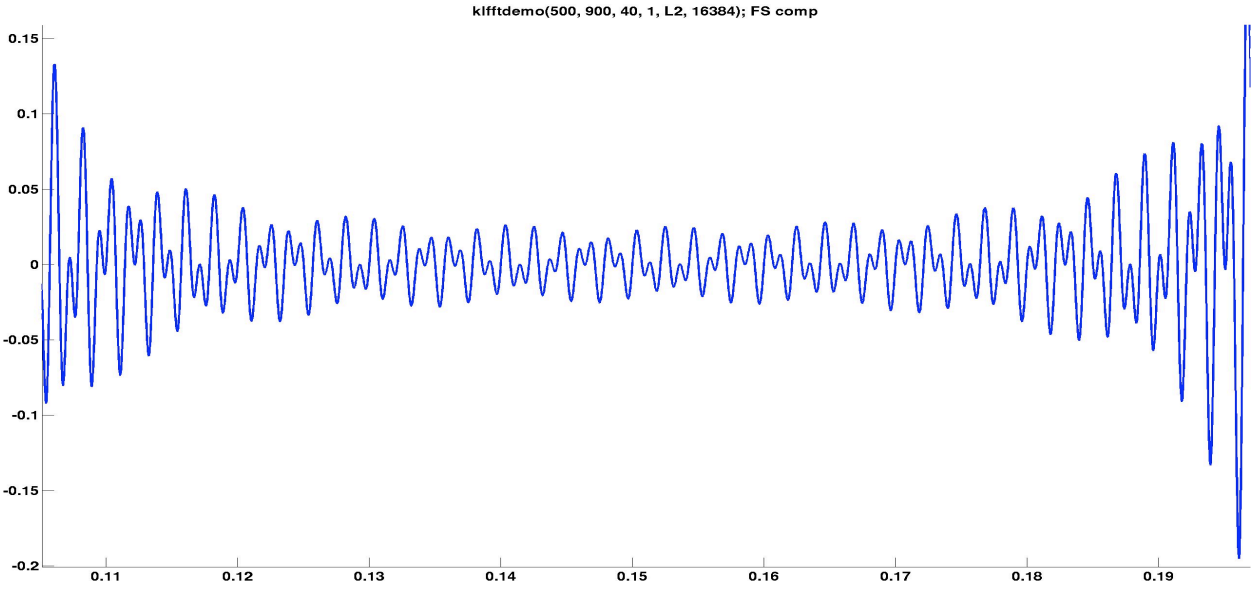


Figure A22 Close-up of Figure A21 Showing Source of Ripple in the Waveform

Appendix 2: Matlab Code

Sample code use:

```
kfd(110, 1.7, 15, 1, 'Q1', 4096) % quadratic spacing of frequencies in freq set
       15 frequencies, start at 110 Hz.
       first frequency delta = (110 / 10) ^ 55
              subsequent delta for frequency n ~ 55 ^ ((n - 1) * 1.7)
       1 sec sample,
       4 Ksample FFT window

% linear spacing of frequencies in frequency set (style 1)
kfd(441, 19, 15, 1, 'L1', 12288)
       15 frequencies, start at 441 Hz.
       constant frequency delta = 19 Hz )
       1 sec sample,
       12 Ksample FFT window

% linear spacing of frequencies in frequency set (style 2)
kfd(1000, 5000, 25, 1, 'L2', 1024)
       25 frequencies, start at 1000 Hz.
       make evenly spaced frequency up to max frequency = 5000 Hz
       1 sec sample,
       1 Ksample FFT window
```

Matlab script kfd.m :

```
function  [ fs ] = kfd(f0, fN, fcount, samplen, fstyle, FFTlen, sweep, sweepstyle)
% demo of some deficiencies in the Cooley-Tukey FFT, and Fourier Analysis
%    1. frequency resolution vs frequency
%
%    f0 = start frequency
%    fN = end freq, or coeff for Q1, E1 etc
%    fcount = % count of frequencies to generate
%    samplen = length of generated audio data in seconds
%      use this to trigger changing freqquencies. every 1.0 sec, fund freq changes
%    fstyle = freq change strategy:
%      L1 = linear -- mult f(n) by a constant: eg f(n+1) = f(n)(1 + delta)
%      L2 = linear -- add a constant to f(n):  eg f(n+1) = f(n) + ndx*delta
%      E1 = exponential going up -- delta f increases exponentially
%      Q1 = quadratic  fn x^pow, delta f increases quadratically
%   future features:
%    sweep = generate smoothly changing freqquencies also
%    sweepstyle = strategy for how sweep change is generated
%
%   fs output == composite waveform of all the generated sinusoids

argc = nargin;
samp_rate = 44100 % CD audio sampling rate per second

% default run params
```

```
f_start = 5000  % fundamental. resolution depends on this one. use Nyquist/Shannon
f_end = 9000
f_count = 20  % generate 20 distinct waveforms
% percent change between freqquencies = (f_end - f_start)/ f_count
samp_len = 1.0 % 1 sec of data == 44100 data points
f_style = 'L1'
fft_len = 256
f_sweep = 0
f_sweepstyle = 0
cycle_factor = 2 * pi % full sine wave at samp_rate (s/b Nyq_freq?, ie 2*pi?)

if argc > 0
    f_start = f0
end
if argc > 1
    f_end = fN
end
if argc > 2
    f_count = fcount
end
if argc > 3
    samp_len = samplen % sample length in seconds
end
if argc > 4
    f_style = fstyle
end
if argc > 5
    fft_len = FFTlen
end
if argc > 6
    f_sweep = sweep
end
if argc > 7
    f_sweepstyle = sweepstyle
end

% generate x grid points, 2 Nyq cycles/sec
x_FS = linspace(0.0, cycle_factor, samp_rate * samp_len)';
FS = zeros(samp_rate * samp_len, f_count + 2); % one row per WF, plus sum total slot
f_list = zeros(f_count + 1, 1);
switch f_style
    case { 'L1' }
        f_coeff = f_end % use fN as delta f
        x_pow = 1
    case { 'L2' }
        f_coeff = (f_end - f_start)/ (f_count ) % N evenly spaced f's between f0 & fN
        x_pow = 1
    case { 'E1' }
        f_coeff = f_end % use fN as delta f
        x_pow = 1
    case { 'Q1' }
        f_coeff = f_start/f_count  %  "evenly" spaced f's (quadratically)
        x_pow = f_end
end
```

```
% freq set construction loop
for runndx = 1:1:f_count + 1  % gives one extra freq than input #
  current_f = f_coeff * (runndx - 1)^x_pow;
  f_list(runndx) = f_start + current_f;
  FS(:, runndx) = FS(:, runndx) + sin((f_start + current_f) * x_FS);
  FS(:, f_count + 2) = FS(:, f_count + 2) + FS(:, runndx); % sum of all harmonics
  FS(:, runndx) = FS(:, runndx) + 2 * ( runndx - 1 ); % for stacking plot
end % freq set construction loop

% norm summed WF to +/- 1
max_amp = max(abs(FS(:, f_count + 2)));
FS(:, f_count + 2) = 2 * (f_count + 2) + FS(:, f_count + 2) / max_amp;

frequencies = f_list % list them here, also below
colors = ['b', 'k', 'r', 'g', 'm']
figure
hold on
% plot the time series waveform. norm to seconds, not samples
plot(x_FS/cycle_factor, FS)
title('FS time series')

% plot a stack of WFs in different colors
%figure
%hold on
%for runndx = 1:1:f_count + 2  % include all component WFs and composite WF
%  % plot the time series waveforms. norm to seconds, not samples
%  plot(x_FS/4*pi, FS(:,runndx) ) % plot each time series waveform
%end

figure
hold on
% plot the composite time series waveform. norm to seconds, not samples
% norm composite WF to +/- 1
plot(x_FS/cycle_factor, FS(:, f_count + 2) - 2 * (f_count + 2), 'LineWidth', 2)
title('FS composite')

 figure
% plot the spectrum vs time (artifacts!)
 specgram(FS(:, f_count + 2), fft_len, samp_rate);

 figure
 plot(f_list)
 title('frequency list')
frequencies = f_list % last visible text output data from script run

% fs = FS;  % return the WF array

end % function nademo(x0, xn, delta, y0, tol)
```

Bibliography

Brigham, O. (1974). ***The Fast Fourier Transform.*** Englewood Cliffs, NJ. Prentice-Hall.

Elliott, Douglas F. & Rao, K. Ramamohan. ***Fast Transforms: Algorithms, Analyses, Applications.*** Academic Press. Orlando, San Diego, San Francisco. 1982.

Hamming, R. W. (1983). ***Numerical Methods for Scientists and Engineers, 2nd ed.*** McGraw-Hill Book Company, New York.

Press, William H., Teukolsky, Saul A., Vetterling, William T., & Flannery, Brian P. (2002) ***Numerical Recipes in c++: The Art of Scientific Computing, 2nd edition.*** Cambridge University Press. Cambridge, UK.

Young, Robert M. ***An Introduction to Nonharmonic Fourier Series, revised 1st ed.*** Academic Press. San Diego, San Francisco. 2001.