Note Frequency $= 440 \times (2^{n/12})$

~~audio source Start~~

Start

stream

onSample
{
if (length > lastlength > fftsize)
{
if (buffer is cleared)
{ lastlength = length
Copy stream to buffer
read
}
else if (stream.length > limit
{ ~~stream.write~~
Clear stream
}
} else stream.write

read

25.419
±12.7
427.474 | 452.893
| 479.824

$$C = (1200)(\log_2(f_2/f_1))$$

or

$$C = (1200)(3.322038403)(\log_{10}(f_2/f_1))$$

$$\delta = Re\left\{\frac{G_{m+1} - G_{m-1}}{2 \cdot G_m - G_{m+1} - G_{m-1}}\right\}$$

$$f_k = \frac{K}{n} f_s$$

$$k = \frac{n}{f_s} \frac{1}{T}$$

$\frac{1}{T}$ = fundamental frequency

λ hz

pos		mag	mag
80	→	.0681	.0046
81	→	.1608	.0259
☆82	→	.4438	.1969
83	→	.0931	.0087
84	→	.052	.0027

$$mag\ \delta = \frac{.0931 - .1608}{2 \cdot .4438 - .0931 - .1608} = \frac{-.0677}{.6337} = -.107 \xrightarrow{+82} k = 81.8932$$

$\cancel{mag = \frac{-.0172}{.3592} = -.0479}$

$$f_k = \frac{k}{8192} 44100$$

$$f_k = 440.8557$$

k = 81.73

fundamental frequency (soundsample x, samprate, min freq, max freq)

1. - Calculate FFT
   ↳ get spectrogram (abs(FFT)^2)
2. - Find peaks in FFT frequency bins
   ↳ based from useful min/max freq
3. - Find peak wit smallest difference in value
4. - Return sample rate % min OptimalInterval

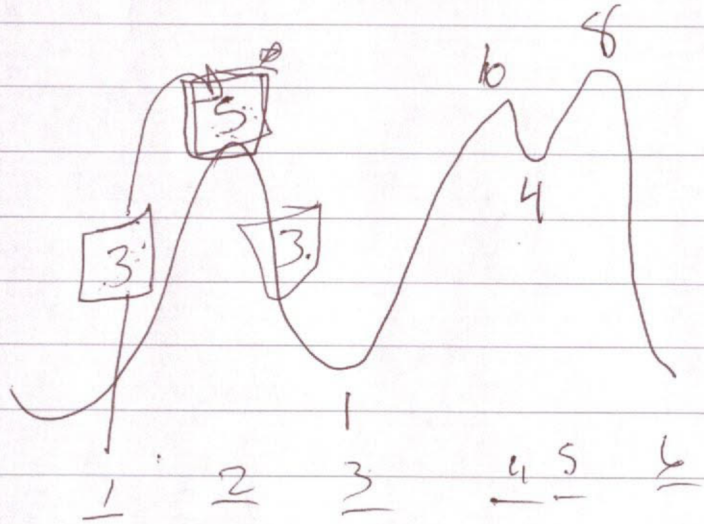
0	Ø	284.42
4	1	120.1
14	8	53
22	10	
213	15	

0 1 2 3 4

~~3 >= 3~~ ✓

~~max=3~~

3 >= 3 ✓
max=3 ✓
3 > 3 X
5 > 3 ✓
max=5
5 > 5 X
3 > 3 X
max = store
1 > 3 X
min=1



min=0

max= ~~X~~, 2

mic

audioin — audio Sample
wrote to memory

* what does this data represent?

Stream in bytes

readinto buffer[]

Convert bytes to
~~double~~[]

Convert ~~bytes~~ # ~~double~~[] to
Complex Number[]

FFT

energy of audio w/in range of freq

- returns array of values (complex Numbers)
- each array member contains the strength of that signal in that frequency band

b/n

Find array member w/ strongest value
to dial in the exact frequency, do some interpolation between the array members around the strongest bucket

24.255.125.98

Class: AudioSink

OnSample →

Stream.Write → (byte) buffer → What does this data Represent?

[0]	43
[1]	1
[2]	158
[3]	240
[4]	235
[5]	215
[6]	220
[7]	132

To double[ ]

a) BitConverter.ToDouble

b) (double) buffer

c) buffer/~~3~~32768.0

2^{16}

Covert to double then complex

stackoverflow: converting-an-fft-to-a-spectogram

FFT/IFFT Of Real Data

fft(v) Returns the Fast Fourier Transform of an n-element vector **v**, where $n = 2^m$. Result is a $1 + 2^{m-1}$ element vector whose jth element is given by:

$$c_j = \frac{1}{\sqrt{n}} \sum_k v_k e^{i(2\pi j/n)k}$$

where n is the number of elements in **v** and i is the imaginary unit. Calculated using the Cooley-Tukey algorithm.

ifft(u) Returns the inverse Fourier transform for **u** created with **fft**. Result is a 2^m element vector whose jth element is given by:

$$c_j = \frac{1}{\sqrt{n}} \sum_k u_k e^{-i(2\pi j/n)k}$$

FFT(v) Returns the Fast Fourier Transform of a vector **v**. The formula is equivalent to **fft**, but is scaled by $1/n$ instead of $1/\sqrt{n}$, and uses a negative exponent going from the time to the frequency domain.

IFFT(u) Returns the inverse Fourier transform for **u** created with **FFT**. The formula is equivalent to **ifft**, but is scaled by 1 instead of $1/\sqrt{n}$, and uses a positive exponent going from the frequency to the time domain.

Arguments:

- **v** is a real-valued vector with 2^m elements $(m > 2)$, representing samples at regular time intervals.
- **u** is a complex-valued vector with $1 + 2^{m-1}$ elements $(m > 2)$, representing samples at frequencies.

Notes:

- The frequency associated with the kth element in the calculated FFT spectrum is given by:

$$f_k = \frac{k}{n} \cdot f_s$$

where f_s is the sampling frequency of the original signal and n is the number of samples. As a consequence, since k must be an integer, spreading in the spectrum occurs unless the sampling frequency is chosen so that

440

$$\frac{n}{f_s} \frac{1}{T} = \text{integer} \rightarrow 81,73$$

is always equal to an integer for any period 1/T in the signal.

- The sampling frequency is distinct from the frequency(ies) of the original time-domain signal.
- The **fft/ifft** functions take advantage of the complex conjugate symmetry of the Fourier transform,

which only applies for real input data. To save time and memory, Mathcad does not calculate the second half of the frequency spectrum. For vectors with complex values, or with an arbitrary number of elements, use **cfft** or **CFFT** instead. Likewise, for 2-dimensional Fourier transforms, use **cfft/CFFT** instead.

- You can also evaluate Fourier transforms symbolically.

QuickSheet

Related Topics