

~~440 Hz~~

Note Frequency = $440 \times (2^{1/12})$



~~Audio source start~~

Start
↓
Stream

on sample Read

{ if (length > lastlength > fft size)

{ if (buffer is cleared)

{ lastlength = length
Copy Stream to buffer

read

}

if (stream.length > limit

~~stream.write~~

Clear stream

}

else stream.write

~~stream~~

25.418

±12.7

427.474

452.893

479.824



Dell EQUALLOGIC

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

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

$$f_k = \frac{k}{n} c_s$$

$$k = n \cdot \frac{1}{f_s} \quad \frac{1}{T} = \text{fundament. f.}$$

$$f_s = \frac{1}{T} = \text{fundamentalfrequenz}$$

24

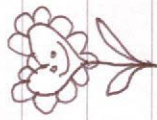
$$\begin{array}{r} 7 \\ 107 \\ \hline 28 \end{array}$$

$$25818$$

[illegible]

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

Fundamental Frequency (\times sample, sample, min freq, max freq)



1. - Calculate FFT

↳ get spectrogram ($\text{abs}(\text{FFT})^2$)

2. - Find peaks in FFT frequency bins

↳ based from useful min/max freq

3. - Find peak with smallest difference in value

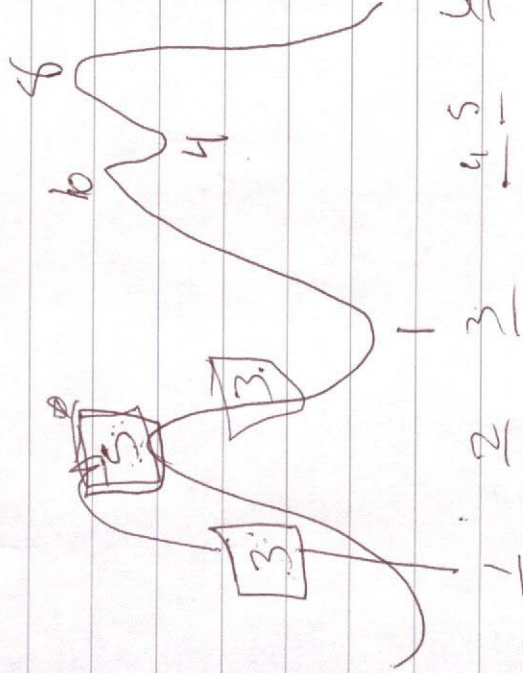
4. - Return sample rate % \times min Optimal Interval

0 284,42
4 120,1
8 53
10
15

0 1 2 3 4

~~3 2 = 3~~

~~max = 3~~



min = 0

max = ~~5~~ 2

3 2 = 3 ✓
max = 3 ✓
3 2 = 3 X
5 2 = 3 max = 5 ✓
5 2 = 3 X

5 2 = 3 X

3 2 = 3 X

max = store

1 2 = 3 X

min = 1

MIC

head

1015

* original

audio in — audio sample write to memory

Stream in bytes

read into buffer[]

convert bytes to

* double[]

* ~~double~~ []

convert to

Complex Number []

convert to double array

FFT

returns array of values (complex numbers)

each member contains the strength of that signal in that frequency band

Find array member with strongest value
to find in the exact frequency, do some
interpolation between the array members around
the strongest bucket

24.255.125.98

class: AudioSink

On sample

Stream: write



(byte)

buffer[0] 43

[1] 1

[2] 158

[3] 240

[4] 235

[5] 215

[6] 220

[7] 132

What does this data request?

To double[]

a) Bit converter. To Double

b) (double) buffer

c) buffer / 52768.0

716

Convert

to double

then complex

Stack overflow - an out-of-memory condition
out-of-memory - an out-of-memory condition
out-of-memory - an out-of-memory condition



EQUALLOGIC

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 $\left(\frac{n}{f_s} \cdot \frac{1}{T} \right) = \text{integer} \Rightarrow 81.73$

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
