Magic Sinewave Quantization Utilities

Don Lancaster
Synergetics, Box 809, Thatcher, AZ 85552

http://www.tinaja.com
don@tinaja.com
(928) 428-4073

A new class of **math functions** called **Magic Sinewaves** lets you efficiently produce power sinewaves that can have **any** chosen number of low harmonics forced very near **zero**. And do so using the fewest possible switching events for the highest possible energy efficiency. Two new intros appear **here** and **here**, along with a development proposal **here**, a tutorial **here**, visualizations **here** , lots of calculators **here**, and seminars and workshops **here**.

In this **Gurugram**, we'll take a closer look at how quantization of magic sinewaves into integer delay and width values affects the synthesis accuracy. In general, quantization introduces errors that dramatically raise distortion levels, creates amplitude jitter, and may introduce frequency jitter. While approximate quantization estimations are provided in the **Magic Sinewave Calculators**, our interests here will be in exact prediction of specific performance.

Some new **PostScript** utilities are available that can give you many of the needed quantization analysis tools. Such as file **OPTJIT01.PSL** and its **OPTJIT01.PDF** demo plots. We'll note in passing that the 32-bit math accuracy of PostScript seems to be acceptable for these specific tasks. As this utility is a personal tool, it may still have some rough edges and may be subject to change and improvement.

Data Formats

The original exact calculators determine a list of first quadrant event positions. In the case of a **Best Efficiency-28 magic sinewave**, there are seven pulses and eight delays whose **absolute** starting points are **0, p1s, p1e, p2s, p2e, p3s, p3e, p4s, p4e, p5s, p5e, p6s, p6e, p7s,** and **p7e**. For a given amplitude, these will be an increasing series of degree values in the 0 to 90 degree range.

As we have seen in our **Magsine Visualizations** as **GuruGram #24**, the situation is somewhat more complex for **delta friendly** magic sinewaves in that half of the pulse edges must track the others to guarantee three phase compatibility. A new **export format** has been added to commercial versions of the calculators that outputs **predelay, p5w, p1w, middelay, p2w, p4w, p3w**, and **postdelay**.

Because of the needed locking, derived widths **p6w** will equal the sum of **p5w** and **p1w**, while **p7w** must equal the sum of **p2w, p4w** and **p3w**.

Further, the **predelay...postdelay** sequence repeats every **thirty** degrees. For any thirty degree interval, the sourcecode will decide which phase gets pulses **p1, p2** and **p3**, which gets **p4** and **p5**, and which gets **p6** and **p7**. Of correct polarity.

The Delta friendly transform from event positions to export format is…

```
predelay = 60 - p5e
p5w = p5e - p5s
p1w = p1e - p1s
middelay = p2s - p1e
p2w = p2e - p2s
p4w = p3s - p2e
p3w = p3e - p3s
postdelay = 30 - p3e
```

And one unquantized reverse transform is…

```
CONTROLLABLE (INDEPENDENT) VARIABLES:

    p1s = predelay + p5w
    p1e = p1s + p1w
    p2s = p1e + middelay
    p2e = p2s + p2w
    p3s = p2e + p4w
    p3e = p3s + p3w
    p5e = 60 - predelay

LOCKED TRACKING (DEPENDENT) VARIABLES:

    p4s = 60 p3s sub
    p4e = 60 p2e sub
    p5s = 60 p1s sub
    p6s = 120 p5e sub
    p6e = 60 p1e add
    p7s = 60 p2s add
    p7e = 60 p3e add
```

While any seven of the fourteen Delta-28 variables could be made independent, the choice of **p1s, p1e, p2s, p2e, p3s, p3e** and **p5e** appears to have some analysis and optimization advantages. Especially when optimizing quantized distortion.

Quantization Tradeoffs

Quantization of magic sinewave data into integer sequences trades off the best achievable distortion against the clock frequency, data word size, amplitude jitter, and possibly frequency jitter. While "reasonable" results can often be gotten by using "modified" 8-bit stored data, optimization methods may not be clear.

A good baseline starting point can be to let a frequency very near 10 Megahertz equal a magic sinewave output frequency of 60 Hertz. If **3472** clock cycles are used in a 30 degree interval, an entire cycle will consist of **41664** cycles . And a 10 MegaHertz PIC clock should produce an output frequency of **60.0038** Hertz. Conversely, a frequency of 60 Hertz can be gotten from a **9.999360** MHz PIC clock. Note that a PIC clock is **four times** the instruction cycle time.

These particular values give us a **2.1198** degree full scale for 8-bit data, which corresponds to one count being **0.008641** degrees. Values above the 2.1198 full scale are easily handled by adding a ramp of **amplitude*k**. Depending on the actual data, inverse ramps and/or end truncated ramps can also be used. Thus, as we have already seen in **MINDIST1.PDF** as **GuruGram #19**, we can get a lot more than 8-bit accuracy and still use 8-bit stored data values.

Quantization consists of converting exact degree values into a group of eight integers that are supposed to sum to 3272 over a 30 degree interval. The degrees to integers algorithm is…

```
degreesperbit = 0.008641
quantout = int ( round (degreesin/degreesperbit))
```

And the corresponding reverse transform is

```
degreesperbit = 0.008641
degreesout = quantin * degreesperbit
```

Thus an amplitude **0.54** array of **[3.929702 6.271940 1.1360694 7.174293 0.5446923 4.7354755 2.7413283 3.466498]** degrees becomes an integer sequence of **[454 727 132 829 63 549 318 400]**. Once again, note that these larger integers can be generated by an 8-bit table lookup plus an 8-bit function.

For instance, the 549 value might be synthesized as an 8-bit stored value of 117 added to 8 times the desired amplitude of 54. Or 117 + (8*54) = 549. Note further that a delay of eight times a stored value gets easily done simply by creating a delay loop of **eight instruction cycles per count**.

A minor gotcha

When we quantize, the "virtually zero" distortion is very likely to get a lot worse. How much worse can be found by converting the integers back into degrees and then recalculating the Fourier coefficients. But, before we do, note that…

The sum of the quantizations does NOT necessarily equal the quantization of the sums!

Each and every 8 data value degree array summed to 30 degrees. But when you quantize, you get a Gaussian type thingy that depends on the roundup or rounddown stats of the individual data. Typically, one-half of your quantized amplitudes will sum to 3472. Most of the others will sum to 3471 or 3473, while a few stragglers will sum to higher or lower values.

This means that…

Direct use of quantized data can cause frequency jitter!

In this example, a frequency jitter of 2 counts out of 3472 translates to just under **0.05** percent jitter worst case. Which is probably negligible for most apps. But we can easily achieve zero jitter with minor mods to our quantized data. At only a negligible distortion penalty.

Ferinstance, you could manually go through the data. If you have a 3471 sum, find the **one** pre rounding value closest **below** 0.5 and round it **up** instead. I automated this in a utility that initially "rounds" everything at a 0.75 threshold, guaranteeing low data. The threshold is then lowered in tiny increments till the 3472 value is reached for each and every quantized data array.

If a quantized array with jitter was, say, **[454 727 132 829 63 549 318 400]**, its frequency dejittered improvement might look like **[454 727 133 829 63 549 318 400]**. I feel that frequency dejittering is most often a very good idea.

Amplitude Jitter

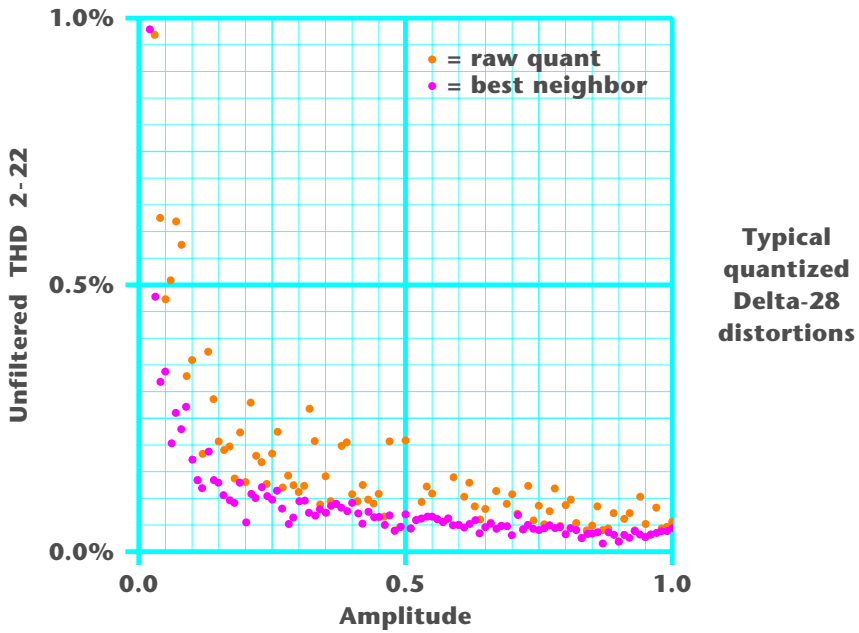
Quantizing will also give us some amplitude jitter, but this is usually not too significant. If you select a hundred amplitude levels, this implies you are willing to take anything from **53.50001** to **54.44449** for amplitude 54 . Which I'll define as an amplitude jitter of plus or minus fifty percent.

As we'll shortly see, purposely allowing a max amplitude jitter around the ten percent range can significantly lower our quantized distortion levels.

Conversely, distortion could be traded off for lower amplitude jitter.

Some Results

The orange dots here give us a preliminary plot of the raw quantization errors we can expect from our above example…



We see that our raw quantized total harmonic distortion for the second through twenty-second harmonic is under 0.2 percent for most higher amplitude values. And that is **before** any filtering. Our filter, of course, has bigger things to worry about. Namely the uncontrolled twenty third and higher harmonics.

We see that 8-bit data thus gives us "acceptable" distortion after quantization in this **10 MHz --> 60 Hz** example. We might be tempted to try and reduce our clock frequency further. Perhaps going to **6 MHz --> 60 Hz** or even using that internal **4 MHz** PIC clock option for fixed 60 Hertz apps. But these reductions may make synthesizing very low or very high amplitudes tricky, especially for Delta 44 or Delta 60 magic sinewave options.

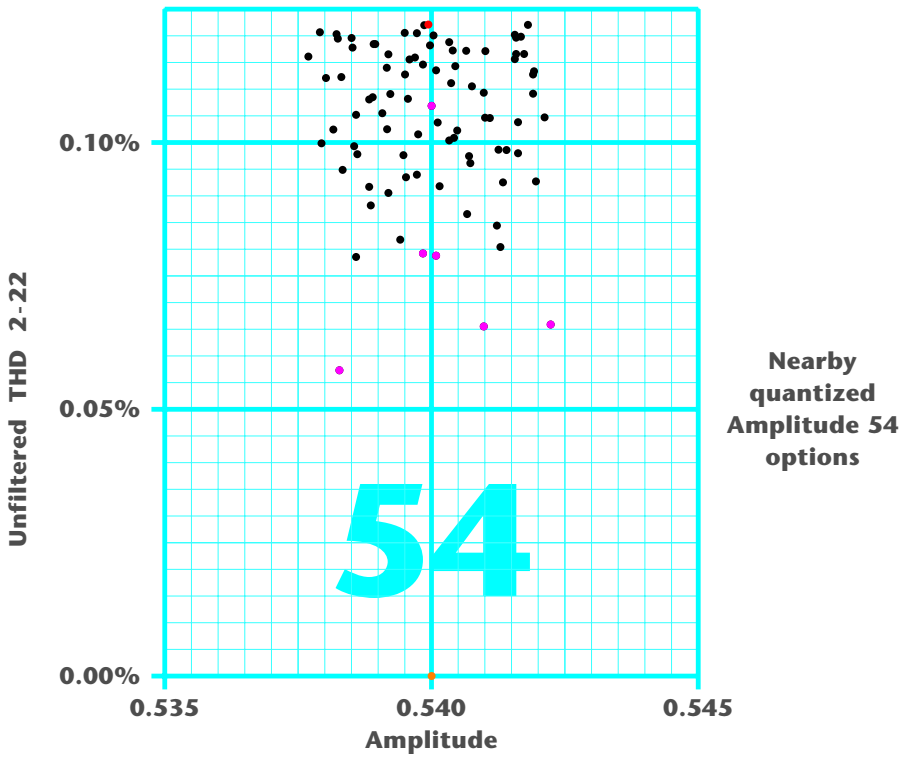
It does turn out there is a sneaky trick we can pull that can further reduce our quantized harmonic distortion. I'll call this one…

Shaking the Box

Compared to our quantized solution, what are the nearest **adjacent** 78,125 magic sinewaves up to? It turns out that a "nearby" magic sinewave can often give us a **two to twelve decibel** thd 2-22 distortion reduction.

Which can result in the magenta dots in our previous figure. Reducing most prefiltered 2-22 distortions to well under one tenth of a percent total thd.

To "shake the box", you take your independent **p1s, p1e, p2s, p2e, p3s, p3e**, and **p5s** integers and alter each value by **-2, -1, 0, 1,** or **2** counts while you are doing an exhaustive search. The results for amplitude 54 might look like this…



I've left those nearby results with higher distortions unplotted. Here, we see that the orange dot is what we want in the way of a "perfect" magic sinewave, the red dot is our initial quantization, and the magenta dots offer several "better" tradeoffs. In this example, I'd go with the 0.0655% x 0.5409 amplitude improved solution.

Note that all of the new data values will still sum to 3472. Because half the delay and position values are dependent on (and calculated from) the other half. No new frequency jitter is introduced by this process.

When plotted on a different scale and allowing higher distortion values, this gives us a classic "dripping stalactite" plot. One that was crucial in early magic sinewave development. When it was found that the drips could be miraculously forced to zero if you used accurate enough calculations.

A Data Set

Here's a preliminary data set using our above example that may be used for sourcecode generation...

```
[ [868   0   0 1730   0   0   0 868] 0.000000 0.0000000 3472 ]
[ [860  13   2 1722   0  10   6 859] 1.510760 0.0093493 3472 ]
[ [850  29   5 1701   6  22   8 851] 0.978672 0.0210324 3472 ]
[ [845  43   7 1683   7  33  14 840] 0.477679 0.0312886 3472 ]
[ [837  56   9 1668   7  43  20 832] 0.318167 0.0406482 3472 ]

[ [830  68  11 1653  10  52  23 825] 0.337353 0.0493669 3472 ]
[ [821  85  14 1631  11  65  30 815] 0.203131 0.0617085 3472 ]
[ [815  95  16 1619  13  73  33 808] 0.259957 0.0692016 3472 ]
[ [807 110  18 1601  14  84  39 799] 0.229666 0.0798052 3472 ]
[ [800 122  20 1587  16  93  43 791] 0.271299 0.0885358 3472 ]

[ [791 138  23 1567  18 106  49 780] 0.172600 0.1005590 3472 ]
[ [784 151  25 1551  19 116  54 772] 0.1339800 0.109939 3472 ]
[ [777 163  27 1536  21 125  58 765] 0.1192960 0.118677 3472 ]
[ [768 180  30 1515  22 138  65 754] 0.1874040 0.131049 3472 ]
[ [761 193  32 1499  24 148  70 745] 0.1343130 0.140698 3472 ]

[ [754 205  34 1484  26 157  74 738] 0.1291950 0.149444 3472 ]
[ [747 218  36 1468  26 167  80 730] 0.1061010 0.158846 3472 ]
[ [738 234  39 1448  28 179  86 720] 0.0965049 0.170589 3472 ]
[ [731 246  41 1434  29 188  91 712] 0.0914561 0.179351 3472 ]
[ [723 260  43 1417  31 199  96 703] 0.1290420 0.189653 3472 ]

[ [714 276  46 1397  32 211 103 693] 0.0546426 0.201417 3472 ]
[ [707 289  48 1381  33 221 108 685] 0.1085760 0.210828 3472 ]
[ [701 300  50 1367  35 229 112 678] 0.1002610 0.218951 3472 ]
[ [691 316  53 1348  36 241 119 668] 0.1207730 0.230727 3472 ]
[ [684 330  55 1330  37 252 125 659] 0.1038410 0.241057 3472 ]

[ [677 342  57 1316  38 261 130 651] 0.0973585 0.249840 3472 ]
[ [670 354  60 1300  40 270 135 643] 0.1141750 0.259094 3472 ]
[ [663 367  62 1284  40 280 141 635] 0.0808775 0.268537 3472 ]
[ [653 384  65 1263  42 293 148 624] 0.0519740 0.281232 3472 ]
[ [647 395  67 1249  43 301 153 617] 0.0641827 0.289386 3472 ]

[ [638 411  70 1229  44 313 160 607] 0.0942731 0.301193 3472 ]
[ [630 424  72 1214  45 323 166 598] 0.0954356 0.310907 3472 ]
[ [624 435  74 1200  46 331 171 591] 0.0730116 0.319070 3472 ]
[ [615 451  77 1180  47 343 178 581] 0.0677774 0.330889 3472 ]
[ [608 463  79 1165  48 352 183 574] 0.0797174 0.339699 3472 ]
```

[[600 477 82 1147 49 363 190 564] 0.0727672 0.350554 3472]
[[592 491 84 1130 50 373 196 556] 0.0859002 0.360616 3472]
[[585 503 87 1114 51 382 202 548] 0.0895733 0.369923 3472]
[[578 516 89 1098 51 392 209 539] 0.0823565 0.379679 3472]
[[569 531 92 1079 53 403 216 529] 0.0758395 0.391137 3472]

[[562 543 94 1064 54 412 221 522] 0.0916116 0.399965 3472]
[[553 558 97 1046 54 423 229 512] 0.0716472 0.411197 3472]
[[547 569 99 1032 55 431 234 505] 0.0527390 0.419389 3472]
[[538 584 102 1013 56 442 242 495] 0.0744991 0.430883 3472]
[[530 598 105 995 56 453 249 486] 0.0643956 0.441523 3472]

[[523 610 107 980 57 462 255 478] 0.0651337 0.450642 3472]
[[515 624 110 962 58 472 262 469] 0.0500311 0.461231 3472]
[[508 636 112 947 59 481 268 461] 0.0682797 0.470357 3472]
[[501 648 115 931 59 490 275 453] 0.0390659 0.479727 3472]
[[493 661 118 914 60 500 282 444] 0.0464662 0.491163 3472]

[[485 675 120 897 61 510 289 435] 0.0701265 0.500383 3472]
[[478 687 123 881 61 519 296 427] 0.0436721 0.509765 3472]
[[470 700 126 864 62 529 303 418] 0.0589122 0.520046 3472]
[[463 712 129 848 62 538 310 410] 0.0621049 0.529436 3472]
[[454 727 132 829 63 549 318 400] 0.0655310 0.540979 3472]

[[447 739 134 814 64 557 324 393] 0.0659652 0.549818 3472]
[[440 751 137 797 64 567 332 384] 0.0608129 0.559810 3472]
[[432 764 140 780 65 577 339 375] 0.0552627 0.570111 3472]
[[424 777 143 764 65 586 346 367] 0.0621947 0.579854 3472]
[[417 789 145 748 66 595 353 359] 0.0497526 0.589296 3472]

[[409 802 149 730 66 605 361 350] 0.0498109 0.599860 3472]
[[401 815 152 713 66 615 369 341] 0.0454799 0.610207 3472]
[[393 828 155 696 67 624 377 332] 0.0516502 0.620493 3472]
[[385 841 158 679 67 634 385 323] 0.0588497 0.630849 3472]
[[379 851 160 665 68 642 391 316] 0.0342060 0.639051 3472]

[[371 864 164 647 68 652 399 307] 0.0456603 0.649639 3472]
[[363 877 167 630 68 662 407 298] 0.0522574 0.660009 3472]
[[355 890 170 612 69 672 415 289] 0.0426991 0.670628 3472]
[[348 901 173 597 69 680 423 281] 0.0480973 0.679712 3472]
[[340 914 176 579 70 690 431 272] 0.0477716 0.690340 3472]

```
[ [333  925 179  564  70  699 438 264] 0.0308037 0.699467 3472 ]
[ [324  939 183  545  70  710 447 254] 0.0686215 0.711012 3472 ]
[ [317  950 185  530  71  718 455 246] 0.0415750 0.720131 3472 ]
[ [310  961 188  514  72  727 462 238] 0.0498212 0.729513 3472 ]
[ [302  973 192  497  71  737 471 229] 0.0430787 0.739846 3472 ]

[ [294  986 195  479  72  747 479 220] 0.0405623 0.750499 3472 ]
[ [286  998 198  462  72  757 488 211] 0.0438831 0.760860 3472 ]
[ [279 1009 201  446  73  766 495 203] 0.0497878 0.770257 3472 ]
[ [271 1021 204  429  74  775 504 194] 0.0441616 0.780553 3472 ]
[ [264 1031 207  414  74  784 512 186] 0.0457397 0.789662 3472 ]

[ [256 1043 211  396  74  794 521 177] 0.0325654 0.800267 3472 ]
[ [249 1054 213  381  75  803 528 169] 0.0440074 0.809451 3472 ]
[ [241 1066 216  363  76  813 537 160] 0.0403837 0.820078 3472 ]
[ [233 1077 220  346  76  823 546 151] 0.0256970 0.830362 3472 ]
[ [225 1089 223  328  77  833 555 142] 0.0332166 0.840997 3472 ]

[ [218 1099 225  313  78  842 563 134] 0.0337785 0.850140 3472 ]
[ [210 1110 228  296  79  852 572 125] 0.0360605 0.860449 3472 ]
[ [203 1120 231  280  80  862 580 117] 0.0150460 0.869889 3472 ]
[ [195 1131 233  262  82  872 589 108] 0.0360463 0.880467 3472 ]
[ [187 1141 236  245  83  883 598  99] 0.0317396 0.890767 3472 ]

[ [180 1151 237  230  85  892 606  91] 0.0190775 0.899933 3472 ]
[ [172 1162 238  213  88  902 615  82] 0.0316686 0.910273 3472 ]
[ [165 1171 239  197  91  912 623  74] 0.0263476 0.919661 3472 ]
[ [157 1181 239  180  95  922 633  65] 0.0388174 0.929956 3472 ]
[ [150 1190 238  164 100  932 641  57] 0.0323680 0.939357 3472 ]
[ [142 1200 235  147 107  943 650  48] 0.0272150 0.949701 3472 ]

[ [134 1209 230  130 116  955 659  39] 0.0318331 0.960034 3472 ]
[ [126 1218 223  113 127  966 669  30] 0.0351219 0.970334 3472 ]
[ [119 1226 210   97 144  977 677  22] 0.0379512 0.979745 3472 ]
[ [111 1234 188   80 170  989 688  12] 0.0386641 0.990337 3472 ]
[ [103 1242 151   63 211 1001 698   3] 0.0439439 1.000650 3472 ]
```

As before, the quantized integer matrix is made up of **predelay, p5w, p1w, middelay, p2w, p4w, p3w, and postdelay**. Followed by the distortion in percent, the normalized amplitude, and the integer count per thirty degrees. The latter also serves as a frequency jitter checksum.

Also as before, values above an 8-bit "full scale" of 255 are realized by summing with an eight bit function. Typical functions are ramps of **amplitude*k**, its inverse, or truncated versions of same.

Some Fourier Math

I overwhelmingly prefer to use the **PostScript** language for all of my engineering analysis. Especially when graphic display is involved. Sadly, PostScript uses only 32-bit math routines making it inappropriate for actual determination of true magic sinewave zero nulls.

Which is why we went to the 64-bit Javascript routines for all the actual **Magic Sinewave Calulators** Typically, **PostScript** has a noise floor in the sixth or seventh decimal place, while JavaScript can easily work fifteen decimal places or more.

Fortunately, **PostScript** is good enough for routine magic sinewave use as we've done here. Here is how you calculate a fundamental amplitude using classic DFT **Fourier Series**…

```
/a1 {curdegarray 0 get cos curdegarray 1 get cos sub
     curdegarray 2 get cos add curdegarray 3 get cos sub
     curdegarray 4 get cos add curdegarray 5 get cos sub
     curdegarray 6 get cos add curdegarray 7 get cos sub
     curdegarray 8 get cos add curdegarray 9 get cos sub
     curdegarray 10 get cos add curdegarray 11 get cos sub
     curdegarray 12 get cos add curdegarray 13 get cos sub
     4 mul pi div} store
```

Here **curdegarray** is **p1s, p1e, p2s, p2e, p3s, p3e, p4s, p4e, p5s, p5e, p6s, p6e, p7s,** and **p7e**. And **pi** is predefined as 3.1415926 or thereabouts.

A typical relative harmonic is calculated as…

```
/h7 {curdegarray 0 get 7 mul cos
     curdegarray 1 get 7 mul cos sub
     curdegarray 2 get 7 mul cos add
     curdegarray 3 get 7 mul cos sub
     curdegarray 4 get 7 mul cos add
     curdegarray 5 get 7 mul cos sub
     curdegarray 6 get 7 mul cos add
     curdegarray 7 get 7 mul cos sub
     curdegarray 8 get 7 mul cos add
     curdegarray 9 get 7 mul cos sub
     curdegarray 10 get 7 mul cos add
     curdegarray 11 get 7 mul cos sub
     curdegarray 12 get 7 mul cos add
     curdegarray 13 get 7 mul cos sub
     4 mul pi div 7 div a1 div} store
```

Other harmonics are suitably scaled, such as multiplying angles by 5 and dividing output by 5 on the fifth harmonic, etc…

Although the triad harmonics of 3, 9, 15, and 21 should be identically zero in a **delta friendly magic sinewave**, it still may pay to calculate them. Both to find possible errors and to keep track of the **PostScript** repeat calc noise floor.

Total harmonic distortion **2-22** is then calculated by…

```
/thd {h3 dup mul h5 dup mul add h7 dup mul add
      h9 dup mul add h11 dup mul add h13 dup mul add
      h15 dup mul add h17 dup mul add h19 dup mul add
      h21 dup mul add sqrt 100 mul} store
```

For More Help

Additional **Magic Sinewave** help is available per the previously shown web links. Detailed analysis of system specifics are available at our standard consulting rates. Per our **Magic Sinewave Development Proposal** and our **Consulting Services**.

Additional **GuruGrams** columns await your ongoing support as a **Synergetics Partner**.