Sine Wave Generation using the Goertzel Algorithm

T080112-00

Daniel Sigg
May 1st, 2008

Discrete Fourier Transform

The Goertzel algorithm is the implementation of a discrete Fourier transform as a inifinite-impulse response filter. Let's consider

$$H(z) = \frac{1}{1 - W\,z^{-1}} \qquad \text{with} \quad W = e^{2\pi i\, f/f_N} \quad \text{and} \quad f_N \text{ the Nyquist frequency.} \tag{1}$$

The time domain representation of this filter is then given by

$$y(n) = x(n) + W\,y(n-1) = \sum_{k=-\infty}^{n} W^{n-k}\,x(k) = W^n \sum_{k=-\infty}^{n} W^{-k}\,x(k) \tag{2}$$

If we set x (k) = 0 for all $k < 0$ and $k = N$, we get

$$y(N) = W^N \sum_{k=0}^{N-1} W^{-k}\,x(k) \tag{3}$$

which corresponds to the discrete Fourier term at frequency f with an additional scale factor W^N. If we want to comnpute the standard discrete Fourier transform, the above expression has to be evaluate for all W of the form $W_k = e^{2\pi i k/N}$ with $k = 0, 1, \ldots, N-1$. In this case we get $W^N = 1$. By setting up filters for all W_k we will get all the terms of a discrete Fourirer transform in an albeit inefficiecient fashion. However, if we are only interested in a single or a few terms, this algorithm is more efficient than computing a full FFT.

Going back to equation (1) this is strictly speaking not an IIR filter, because it consists of a single complex pole. We can fix this by multiplying by $\left(1 - W^* z^{-1}\right)$ both in the denominator and the numerator

$$H(z) = \frac{1 - W^* z^{-1}}{1 - 2\cos(2\pi\, f\,/\,f_N)\,z^{-1} + z^{-2}}. \tag{4}$$

Now we have a normal two-pole filter followd by a complex zero. In the time domain this corresponds to

$$\begin{aligned} v(n) &= 2\cos(2\pi\, f / f_N)\, v(n-1) - v(n-2) + x(n) \quad \text{and} \\ y(n) &= v(n) - W^*\, v(n-1). \end{aligned} \tag{5}$$

The key insighth is that the first line corresponds to a second-order filter section with fixed real coeffiecients. However, the second line which involves a complex multiplication only needs to be computed once at the very end for $y(N)$.

The second line of equation (5) employs a complex coffient because we are interested in the complex Fourier coeffcient. If we were to look for the real part, the imaginary part or the amplitude along a single direction ψ in the complex plane only, we can rewrite equation (5) in the usual direct form II structure

$$\begin{aligned} v(n) &= x(n) + a_1\, v(n-1) + a_2\, v(n-2) \\ y(n) &= b_0\, v(n) + b_1\, v\,(n-1). \end{aligned} \tag{6}$$

Using

$$H_{\text{sos}} = \begin{pmatrix} 1 & a_1 & a_2 \\ b_0 & b_1 & b_2 \end{pmatrix} \tag{7}$$

we get

$$H_{\text{sos}}(\text{Re}) = \begin{pmatrix} 1 & 2\cos(2\pi\, f / f_N) & -1 \\ 1 & -\cos(2\pi\, f / f_N) & 0 \end{pmatrix}, \tag{8}$$

$$H_{\text{sos}}(\text{Im}) = \begin{pmatrix} 1 & 2\cos(2\pi\, f / f_N) & -1 \\ 0 & \sin(2\pi\, f / f_N) & 0 \end{pmatrix} \text{ and} \tag{9}$$

$$H_{\text{sos}}(\psi) = \begin{pmatrix} 1 & 2\cos(2\pi\, f / f_N) & -1 \\ \cos(\psi) & -\cos(2\pi\, f / f_N + \psi) & 0 \end{pmatrix}. \tag{10}$$

Two things are noteworthy. First, in implementations of second-order filter sections the term b_0 is often pulled out as an overall gain factor by setting it to 1. This is not possible here when we are looking for the imaginary part. We can avoid this by using

$$H_{\text{sos}}(\text{Im}) = \begin{pmatrix} 1 & 2\cos(2\pi\, f / f_N) & -1 \\ \sin(2\pi\, f / f_N) & 0 & 0 \end{pmatrix} \tag{11}$$

instead, and looking at $y(n-1)$ rather than $y(n)$. Secondly, one might also be interested in the absolute value or the phase angle. This is straight forward to compute but will no longer constitute a linear filter. For example,

$$\begin{aligned} r(n) &= \sqrt{v(n-1)^2 - 2\cos(2\pi\, f / f_N)\, v(n-1)\, v(n-2) + v(n-2)^2} \quad \text{and} \\ \varphi(n) &= \arctan\left(\frac{\sin(2\pi\, f / f_N)\, v(n-2)}{v(n-1) - \cos(2\pi\, f / f_N)\, v(n-2)} \right) \end{aligned} \tag{12}$$

yield the absolute value and complex phase angle, respectively. Of course, the arctan function needs to take into account in which quadrant the complex number is in.

Sine Wave Generation

The Goertzel algorithm can also be used to generate a sine wave. Consider the input series $x(0) = 1$ and $x(k) = 0$ for all $k \neq 0$, we can see that equation (2) becomes simply

$$y(n) = W^n = e^{2\pi i\, n\, f / f_N} \tag{13}$$

Taking the real part we get a pure cosine function, whereas the imaginary part yields the corresponding sine function.

One problem which will occure for any real world implementation is that numerical round-off errors can accumulate over time. If the desired sine wave has an integer number of cycles within M samples, the filter can simply be restarted every M samples. This avoids the accumulation of numerical errors and the algorithm will give predictable results for any length of running.

If we are interested in a sine wave only, we can used $H_{\text{sos}}(\text{Im})$ from equation (11) and pull the factor b_0 into the input series. We then get $y(n) = v(n)$ and the filter reduces to a complex pole only. To start the sine wave at zero, we restart the algorithm at $n = l\,M$ with $l = 0,\ 1,\ 2,\ \ldots$, we set the $y(l\,M) = 0$ and use the following input series

$$x(k) = \sum_{l=-\infty}^{\infty} \delta_{k,l\,M+1} \sin(2\,\pi\ f\ /\ f_N) \tag{14}$$

with δ the Kronecker delta. This later algorithm has the advantage that the $v(n)$ are always contained within -1 and $+1$. It is therefore suitable for implementations which use fixed point number arithmetic. For a fixed point implementation of a cosine function we need a different approach. Consider the trigonometric series $\sin(\alpha_0 + n\,\alpha)$ and $\cos(\alpha_0 + n\,\alpha)$ with n = 0, 1, ..., $N - 1$. We make the ansatz

$$\begin{aligned} y_s(n) &= \sin(\alpha_0 + n\,\alpha) = x \sin(\alpha_0 + (n-2)\,\alpha) + y \sin(\alpha_0 + (n-1)\,\alpha) \\ y_c(n) &= \cos(\alpha_0 + n\,\alpha) = p \cos(\alpha_0 + (n-2)\,\alpha) + q \cos(\alpha_0 + (n-1)\,\alpha) \end{aligned} \tag{15}$$

Using the trigonometric formulae for expanding sine and cosine of sums we find

$$x = p = -1 \ \text{ and } y = q = 2\cos(\alpha) \tag{16}$$

Again, this corresponds to a simple second order filter section with a complex pole pair. The trick is to load the correct initialization coefficients into $y(-1)$ and $y(-2)$. However, this is fairly straight forward

$$\begin{aligned} y_s(-1) &= \sin(\alpha_0 - \alpha) \quad \text{and} \quad y_s(-2) = \sin(\alpha_0 - 2\,\alpha), \\ y_c(-1) &= \cos(\alpha_0 - \alpha) \ \text{ and } \ y_c(-2) = \cos(\alpha_0 - 2\,\alpha). \end{aligned} \tag{17}$$

Of course, if both sine and cosine functions are required the CORDIC (coordinate rotational digital computer) algorithm could be used as well. The CORDIC algorithm is not of the form of our second-order filter from above, but can be looked at as a two-input two-output IIR filter. Let's look at the following trigonometric equation

$$\boldsymbol{y}(n) = \begin{pmatrix} \cos(\alpha_0 + n\,\alpha) \\ \sin(\alpha_0 + n\,\alpha) \end{pmatrix} = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix} \boldsymbol{y}(n-1) \tag{18}$$

Initializing it with

$$\boldsymbol{y}(-1) = \begin{pmatrix} \cos(\alpha_0 - \alpha) \\ \sin(\alpha_0 - \alpha) \end{pmatrix} \tag{19}$$

will lead to a simultanous generation of sine and cosine functions through coordinate rotation. This algorithm can be expanded to include an input series

$$\boldsymbol{y}(n) = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix} \boldsymbol{y}(n-1) + \boldsymbol{x}(n). \tag{20}$$

This later algorithm can be initialize every M samples by setting all history values to zero. The following time series can be fed into the algorithm to generate cosine and sine wave outputs of amplitude A

$$x(k) = A \sum_{l=-\infty}^{\infty} \begin{pmatrix} \delta_{k,l\,M} \\ 0 \end{pmatrix}. \tag{21}$$

Demodulation

Demodulating a signal with a sine or cosine function and then taking an average is very similar in concept to computing a discrete Fourier coefficient. The descrite Fourier transform from above will compute the cosine and sine terms on time series of fixed length. By virtue of summing over all samples an average is formed. The disadvantage of this algorithm is that it needs to be restarted for every new time series. We prefer to have an algoithm which works continuously. We can write a demodulation as

$$u(k) = W^{-k}\, x(k). \tag{22}$$

Then, we apply a single pole filter with exponential decay

$$y(n) = r\, y(n-1) + (1-r)\, u(n) = (1-r) \sum_{k=-\infty}^{n} r^{n-k}\, u(k). \tag{23}$$

Combining the two equations yields

$$y(n) = W^{-n}(1-r) \sum_{k=-\infty}^{n} r^{\,n-k}\, W^{n-k}\, x(k). \tag{24}$$

Unfortunately, this algorithm generates an output series which is not really demodulated. An adidional rotation by W^n would be required. If we also require a decimation by M and if we can adjust the demodulation frequency so that $W^M = 1$, then the series $y(k\,M)$ will be the demodulated and filterd complex output. Only a single pole filter can be implemented in a straight forward manner, so.

A more flexible approach is to used a cosine/sine generator and multiple the input time series by both of them to form a in-phase and a quad-phase signal. Then, filters of any order can be applied to each phase independently. These two time series can be thought of as a complex time series. They can easily be combined to adjust the demodulation phase, or to compute the absolute value and phase angle.

Equations

GoertzelBasic[l, w] takes a time series l and applies the basic Gortzel algorithm without the complex zero. It can be used to generate a sine wave. w denotes the angular frequency $2\,\pi\, f\, /\, f_N$.

```
GoertzelBasic[l_List, w_] := Block[
  {old = 0, oldold = 0, i, new, res = {}},
  For[i = 1, i <= Length[l], ++i,
   new = 2 Cos[w] old - oldold + l[[i]];
   AppendTo[res, new];
   oldold = old;
   old = new;
  ];
  res
 ]
```

Goertzel[l, w] takes a time series l and applies the Gortzel algorithm. Its output is complex and it can be used to compute a discrete Fourier coefficent. *w* denotes the angular frequency $2\pi f / f_N$.

Goertzel[l, w, r] takes a time series l and applies the Gortzel algorithm incuding an additional pole. Its output is complex and it can be used to form a demodulated filter signal. *w* denotes the angular frequency $2\pi f / f_N$. *r* denotes the filter coefficient of the averaging pole.

```
Goertzel[l_List, w_] := Block[
  {old = 0, oldold = 0, i, new, res = {}},
  For[i = 1, i <= Length[l], ++i,
   new = 2 Cos[w] old - oldold + l[[i]];
   AppendTo[res, new - E^(-I w) old];
   oldold = old;
   old = new;
  ];
  res
 ]
Goertzel[l_List, w_, r_] := Block[
  {old = 0, oldold = 0, i, new, res = {}},
  For[i = 1, i <= Length[l], ++i,
   new = 2 r Cos[w] old - r^2 oldold + l[[i]];
   AppendTo[res, (1 - r) (new - r E^(-I w) old)];
   oldold = old;
   old = new;
  ];
  res
 ]
```

Cordic[l, w] takes a 2-dimensional time series and applies the cordic algorithm using a simple rotation with each step. *w* denotes the angular frequency $2\pi f / f_N$. The output is a two-dimensional series which can be used to generate a cosine and sine wave simultanously.

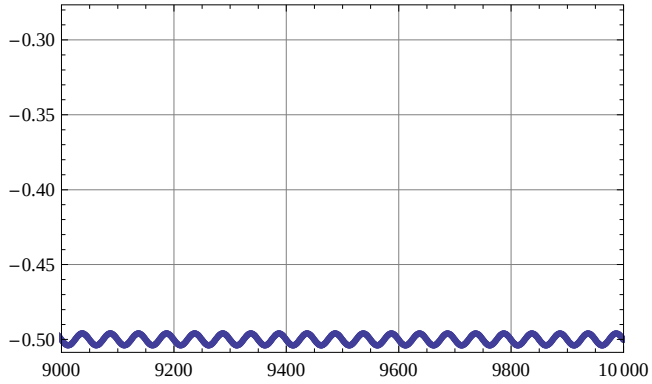
In[770]:=
```
Cordic[l_List, w_] := Block[
  {old = {0, 0}, i, new, res = {}},
  For[i = 1, i <= Length[l], ++i,
   new = {{Cos[w], -Sin[w]}, {Sin[w], Cos[w]}}.old + l[[i]];
   AppendTo[res, new];
   old = new;
  ];
  res
 ]
```

Tests

In[777]:=
```
nn = 10 000;
np = 100.;
rr = 1 - 0.001;
mod = Table[Exp[ i 2 π i / np] , {i, nn}];
uvec = Table[Exp[ i 2 π i / np + i π] , {i, nn}];
g = Goertzel[Re[uvec], 2 π / np, rr];
h = g mod*;
gg = Pick[g, Table[If[Mod[i, 100] == 0, True, False], {i, nn}]];
```

In[785]:=
```
ListPlot[Re[h], Frame → True, GridLines → Automatic, PlotRange → {{9000, 10 000}, Automatic}]
```

Out[785]=



In[786]:=
```
uvec = Table[If[i == 2, Sin[ω], 0], {i, 10}];
g = GoertzelBasic[uvec, ω] // TrigReduce
```

Out[787]=
$$\{0, \text{Sin}[\omega], \text{Sin}[2\omega], \text{Sin}[3\omega], \text{Sin}[4\omega], \text{Sin}[5\omega], \text{Sin}[6\omega], \text{Sin}[7\omega], \text{Sin}[8\omega], \text{Sin}[9\omega]\}$$

In[788]:=
```
uvec = Table[If[i == 1, 1, 0], {i, 10}]; gC = Goertzel[uvec, ω] // TrigReduce
```

Out[788]=
$$\left\{1, e^{i\omega}, e^{2i\omega}, \text{Cos}[3\omega] + i\,\text{Sin}[3\omega], \text{Cos}[4\omega] + i\,\text{Sin}[4\omega], \text{Cos}[5\omega] + i\,\text{Sin}[5\omega], \right.$$
$$\left. \text{Cos}[6\omega] + i\,\text{Sin}[6\omega], \text{Cos}[7\omega] + i\,\text{Sin}[7\omega], \text{Cos}[8\omega] + i\,\text{Sin}[8\omega], \text{Cos}[9\omega] + i\,\text{Sin}[9\omega]\right\}$$

In[789]:=
```
uvec = Table[If[i == 1, 1, 0], {i, 10}];
g = Re[Goertzel[uvec, ω]] // ComplexExpand // TrigReduce
```

Out[790]=
$$\{1, \text{Cos}[\omega], \text{Cos}[2\omega], \text{Cos}[3\omega], \text{Cos}[4\omega], \text{Cos}[5\omega], \text{Cos}[6\omega], \text{Cos}[7\omega], \text{Cos}[8\omega], \text{Cos}[9\omega]\}$$

In[791]:=
```
uvec = Table[If[i == 2, 2 Cos[ω],
    If[i == 3, -1, 0]], {i, 10}];
g = Re[Goertzel[uvec, ω]] // ComplexExpand // TrigReduce
```

Out[792]=
$$\{0, 2\,\text{Cos}[\omega], \text{Cos}[2\omega], \text{Cos}[3\omega], \text{Cos}[4\omega], \text{Cos}[5\omega], \text{Cos}[6\omega], \text{Cos}[7\omega], \text{Cos}[8\omega], \text{Cos}[9\omega]\}$$

In[793]:=
```
vvec = Table[If[i == 1, {1, 0}, {0, 0}], {i, 10}];
Cordic[vvec, ω] // TrigReduce
```

Out[794]=
```
{{1, 0}, {Cos[ω], Sin[ω]}, {Cos[2 ω], Sin[2 ω]}, {Cos[3 ω], Sin[3 ω]},
 {Cos[4 ω], Sin[4 ω]}, {Cos[5 ω], Sin[5 ω]}, {Cos[6 ω], Sin[6 ω]},
 {Cos[7 ω], Sin[7 ω]}, {Cos[8 ω], Sin[8 ω]}, {Cos[9 ω], Sin[9 ω]}}
```