Fast and low-complexity atan2(a,b) approximation

Vicente Torres
vtorres@eln.upv.es
Universitat Politecnica de Valencia

Javier Valls
jvalls@eln.upv.es
Universitat Politecnica de Valencia

Richard Lyons
r.lyons@ieee.org
Besser Associates
Mt. View, California

INTRODUCTION

This article presents a new entry to the class of published algorithms for the fast computation of the arctangent of a complex number. Our method uses a look-up table (LUT) to reduce computational errors. We also show how to convert a large sized LUT addressed by two variables to an equivalent-performance smaller sized LUT addressed by only one variable. In addition, we demonstrate how and why the use of follow-on LUTs applied to other simple arctan algorithms produce unexpected and interesting results.

The computation of the arctangent function atan2(a,b), *i.e.* obtaining the angle of a complex number c=b+ja, has been the subject of extensive study because this computation is needed in many applications. For example, in the frequency, phase, and time synchronization stages of digital communications, digital FM demodulation, target tracking in wireless sensor networks, and object recognition in the field of image processing. From a designer's point of view it is useful to have several computation choices, since the performance requirements (speed, accuracy, power consumption, etc.) may be different depending on the specific application, and one of those choices may be better suited than others for a given application.

A high-speed computation of atan2(a,b) can be achieved with look-up tables (LUTs), where the bit-level concatenation of a and b are the values used to address the ROM that stores the output of the function. The look-up table method is fast but much memory is required when a decent arctangent accuracy is needed. Another popular option is to use high-order algebraic polynomials, like Chebyshev polynomials or Taylor series [1]. These methods give good precision, but since the arctangent is highly non-linear they lead to long polynomials and intensive computations. In other cases, approximations based on rational functions are used [2–4], as they may provide acceptable results with few computations. The coordinated rotation digital computer (CORDIC) algorithm, which requires only shift and add operations, is frequently used to compute the arctangent [1]. However, its sequential nature makes it less adequate when throughput speed is critical.

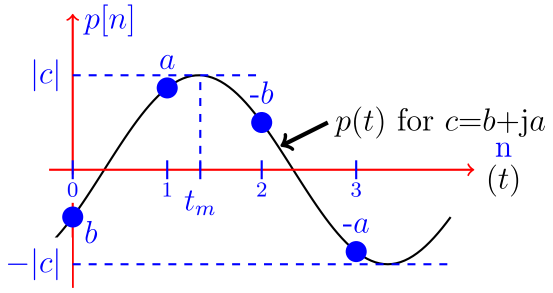
Instead of using a single complicated equation to achieve high accuracy, as proposed by other authors, our proposal is a two-stage process, with a first stage that uses a low-complexity coarse approximation; and a second stage which improves the accuracy by means of a small LUT that stores precomputed error values (as a function of the first stage output). Our proposal computes a full-quadrant arctangent faster than other popular options that achieve the same accuracy. We now describe the two processing stages of our proposed atan2(a,b) algorithm.

FIRST STAGE

The idea behind this stage is to conceptually generate a continuous real-valued sinusoid $p(t)$ that has the same initial phase angle as the phase of our complex number c=b+ja. If $c = |c|e^{j\theta}$, that sinusoid would be:

$$p(t) = |c| \cdot \cos\left(\frac{2\pi t}{T} - \theta\right), \tag{1}$$

where t is time and T is the sinusoid's period, as shown in Figure 1.



[FIG1] Real-valued sequence $p[n]$ and continuous sinusoid $p(t)$ associated with a given complex number c=$|c|e^{j\theta}$.

The reason we care about this $p(t)$ sinusoid is that the time location of $p(t)$'s maximum value, t_m in Figure 1, is proportional to the desired phase angle of c=b+ja = $|c|e^{j\theta}$. The relationship between t_m and θ is found by setting the time derivative of $p(t)$ equal to zero and solving for t_m. Doing so gives us:

$$t_m = \frac{T\theta}{2\pi}. \tag{2}$$

The time-domain dimensions of variables t_m and T must, of course, be identical. With no loss in generality, and out of convenience, we assume the time between the $p[n]$ samples is unity. Thus t_m is measured in units and T=4 units. So when we use (2) to compute θ the ratio t_m/T will be dimensionless and θ will be measured in radians. In the First Stage processing we will estimate t_m and show how this will yield an estimate of angle θ, for a given accuracy, more efficiently than performing other atan2(a,b) algorithms.

It can easily be seen that the sequence $p[n] = \{b, a, -b, -a\}$ is a sampled version of $p(t)$, as defined by (1) as:

$$\begin{aligned} p[n] &= \{b, a, -b, -a\} = |c| \cdot \{cos\left(\theta\right), sin\left(\theta\right), -cos\left(\theta\right), -sin\left(\theta\right)\} \\ &= |c| \cdot \left\{cos\left(-\theta\right), cos\left(\frac{\pi}{2} - \theta\right), cos\left(\frac{2\pi}{2} - \theta\right), cos\left(\frac{3\pi}{2} - \theta\right)\right\} \\ &= |c| \cdot cos\left(\frac{n\pi}{2} - \theta\right) = p\left(t\right)\big|_{t=\frac{nT}{4}} \qquad where\ n = 0, 1, 2, 3. \end{aligned} \tag{3}$$

Our goal is to compute t_m from the $p[n]$ samples.

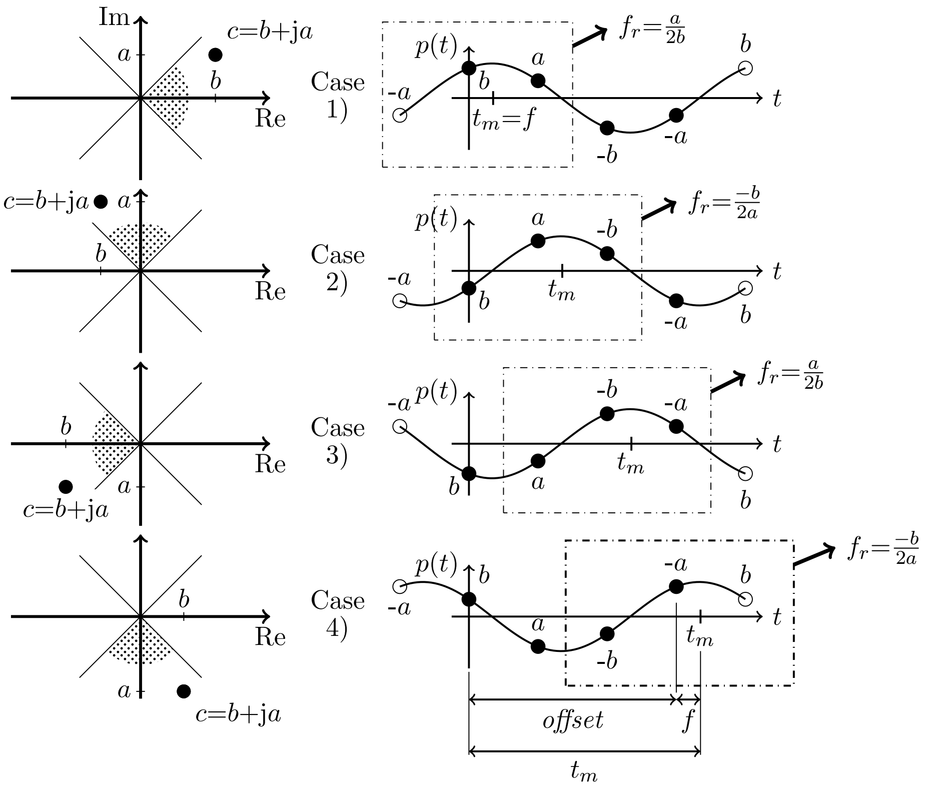
To clarify our scenario here, Figure 2 shows the various $p(t)$ waveforms that result from various values of our complex number input c. The time location of the absolute maximum of the sinusoidal $p(t)$ waveform, t_m, is proportional to the angle of c.

The first step of the First Stage processing is to determine the time location of the largest sample of four-sample sequence $p[n]$ (determined from the signs of a+b and a−b), a parameter that we call "*offset*". The second step of the First Stage processing computes the time location of the maximum value of $p(t)$ relative to *offset*, a parameter that we call "f".

Given the above concepts and relationships, we conclude the first step of the First Stage processing by determining the value for *offset*, which will be 0, 1, 2, or 3. In the second step of the First Stage processing we complete the estimation of t_m approximating the value of time variable f. Specifically we approximate the $p(t)$ signal by a second-order Taylor series in the vicinity of the largest $p[n]$ sample as detailed in the APPENDIX, which gives us f_r, an approximation of the time location of the maximum value of $p(t)$ relative to that sample:

$$f \approx f_r \equiv \frac{-p'(0)}{p''(0)} = \frac{p(1)}{2p(0)} \tag{4}$$

In a general case, this computation would require three samples: the biggest of the four samples of the waveform, and also the two samples adjacent to that sample, as depicted for Case 1) in Figure 2, but since those two adjacent samples have the same absolute value and opposite sign, only two samples are required in (4): the largest sample $p(0)$ and its following



[FIG2] The proposed atan2(a,b) algorithm illustrated for 4 different possible c=b+ja values.

sample $p(1)$. Using (4) we compile our desired processing parameters in Table 1. (Note that a negative value of f_r indicates that $p(t)$ maximum value occurred prior to the largest sample in $p[n]$.)

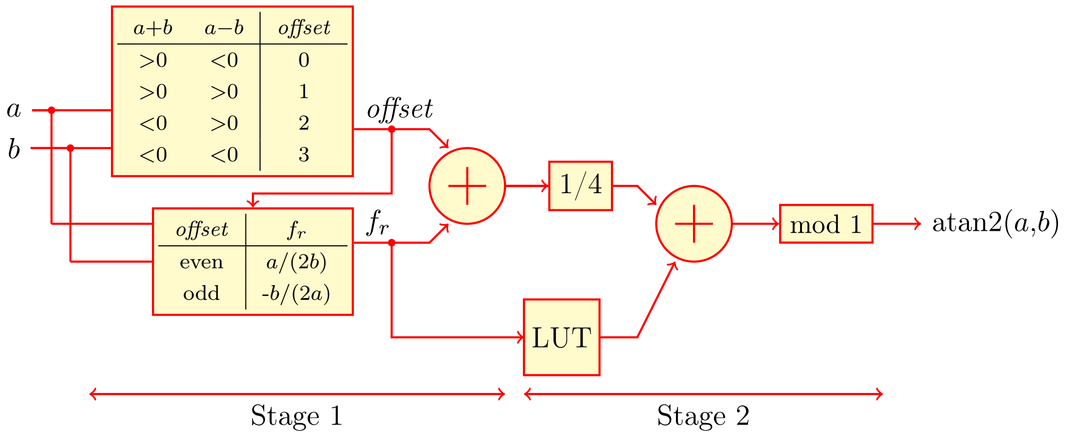
[TABLE 1] Deduction of the expression for f_r as a function of the signs of $a{+}b$ and $a{-}b$.

Case	$a{+}b > 0$	$a{-}b > 0$	$p(0)$	$p(1)$	*offset*	$f_r = \frac{p(1)}{2p(0)}$
1)	1	0	b	a	0	$\frac{a}{2b}$
2)	1	1	a	-b	1	$\frac{-b}{2a}$
3)	0	1	-b	-a	2	$\frac{a}{2b}$
4)	0	0	-a	b	3	$\frac{-b}{2a}$

Based on the values for *offset* and f_r from Table 1 and using (2), assuming T=4, the result of the First Stage processing is an approximation of atan2(a,b), normalized to the range $[0, 1)$, as follows:

$$\frac{atan2(a,b)}{2\pi} = \frac{\theta}{2\pi} = \frac{t_m}{4} \quad mod\ 1 \approx \frac{offset + f_r}{4} \quad mod\ 1 \tag{5}$$

where the mod operator is needed to translate negative values to the desired $[0, 1)$ range. The computation of parameters *offset* and f_r are shown as the First Stage processing in Figure 3.



[FIG3] Building blocks for the proposed atan2(a,b) algorithm.

The neat trick of our proposed algorithm is that neither the $p[n]$ sequence nor the continuous $p(t)$ signal need to be computed. Our First Stage processing produces a rough estimate of the angle of c=b+ja based upon some simple logic and simple arithmetic using values a and b. The *offset* can be computed using the signs of $a{+}b$ and $a{-}b$, as shown in Table 1. The determination of the two samples needed for the computation of f_r can also be performed using the signs of $a+b$ and $a-b$. It should be pointed out that the variable used in the denominator in both expressions for f_r is always the largest absolute value between a and b.

It should be noted that in [5], Shima used an approximation for the one-variable atan(x) (derived from a first order Lagrange polynomial interpolation of the function in two octants) that would lead to the same expression as ours assuming our a/b or b/a were substituted for his x and his expression is extended to the four quadrants using trigonometric identities.

The coarse approximation of atan2(a,b) in this First Stage can be modeled using the following MATLAB-style code, which returns a normalized $\theta/(2\pi)$ value for the arctangent:

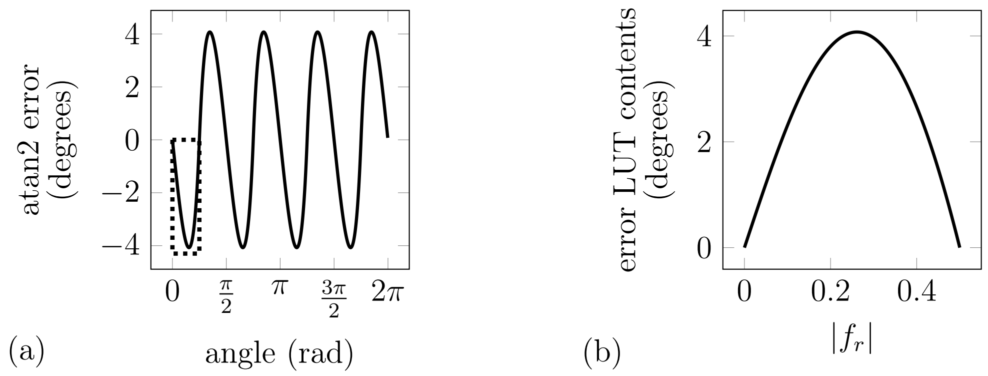
```
function angl=ap_atan2(a,b)
    b0=(a+b)>0;
    b1=(a-b)>0;
    offset=2*not(b0)+not(xor(b1,b0));
    if b0==b1
        fr=-0.5*b/a;
    else
        fr=0.5*a/b;
    end
    angl=mod((offset+fr)/4,1);
end
```

SECOND STAGE

The normalized error obtained using the First Stage is shown in Figure 4(a), as a function of the actual angle of the complex number c=b+ja.

This error function should not be directly stored in a LUT, as it needs to be addressed by the concatenation of the inputs a and b, requiring a large amount of storage. However, we use the trick of transforming that error function to one that only depends on the coarse approximation calculated in the previous stage. Therefore, our proposed Second Stage improves the accuracy of the First Stage result using an error LUT addressed by $|f_r|$.

The MATLAB function 'errorLUTcontents' indicates how this two-variable to one-variable addressing transformation is done where the absolute value of the error is as a function of the absolute value of variable f_r.



[FIG4] Error curves: a) First Stage arctangent error in degrees; b) contents of the Second Stage error LUT.

```
function  f2=errorLUTcontents(wLUT);
    x=linspace(0,0.5,2^wLUT);
    t=linspace(0,pi/4,2^wLUT);
    c=exp(1j*t);
    f1=atan2(imag(c),real(c))/(2*pi);
    f2=0.125*abs(imag(c))./abs(real(c));
    error=f1-f2;
    f2=interp1(4*f2,error,x,'pchip');
end
```

The contents of a LUT named table which contains 2^{wLUT} words can be computed using:

```
table=errorLUTcontents(wLUT);
```

In function 'ap_atan2', right after f_r is computed, the following two sentences would be used to include the Second Stage in the model:

```
    fix=sign(fr)*table(1+floor(abs(fr)*2^(wLUT+1)));
    angl=mod(angl+fix,1);
```

It should be noted that if this method were implemented using finite precision arithmetic, the least-significant bit of the table would be around 3 positions lower than the target accuracy desired for the whole operator.

Summarizing, our complete algorithm to approximate the angle of a complex number is to: 1) identify the maximum of the four $p[n]$ samples in Figure 1 to determine the value of *offset*, 2) compute the time location f_r of the $p(t)$'s maximum and combine that f_r with *offset* value from step 1), and 3) improve the result of step 2) using a relatively small error LUT.

RESULTS AND PERFORMANCE

In this section we will compare our approach to several known low-complexity approximations of the atan2 function.

Table 2 summarizes the computational resources needed by our proposal and a few other atan2 approximations, grouped for akin accuracies. Whenever an algorithm is proposed for a two-octant one-variable atan(x), three extra addition/subtractions are included in this table as the cost of extending the approximation to all the quadrants. In Table 2 we only consider divisions, multiplications, additions/subtractions and storage requirements to evaluate the computational cost of the algorithms. Other required operations have a computational cost that can be highly platform-dependent. Operators like binary shifts, mod(), floor(), bit string concatenations, etc. have no cost at all in fixed-point ASIC or FPGA implementations, but may result in additional computational time in pure software implementations. For example, in a fixed-point FPGA implementation the computation of offset =2∗not(b0)+not(xor(b1,b0)) doesn't involve multiplications nor additions, just bit concatenations and simple logic operators. In such a case, the hardware architecture could be implemented following the data flow illustrated in Figure 3.

When only the First Stage of the algorithm, *i.e.* without the error LUT, is used, the approximation of the atan2(a,b), has a maximum error of $\pm 4.07°$, which corresponds to 6.5 exact bits (*i.e.* the number of most significant bits that are zero in the binary representation of the maximum absolute value of the error; this value can be obtained as $-\log_2(\max(\text{abs}(error)))$, assuming a $[0, 1)$ normalization of the values). Another coarse approximation was presented in [6]-(Eq. 12), achieving the same accuracy with higher computational cost.

As shown in Table 2, by using a small error LUT of 32 values our proposal achieves similar accuracy to Lyons' [2]-(Eq. 2) and Rajan *et al.*'s [3]-(Eq. 9), but requiring three and two fewer multiplications, respectively. If larger LUT sizes are used, similar accuracy to [4]-(Eq. 18 and Eq. 16) can be achieved with fewer arithmetic resources by our method.

Finally, another option considered for comparison purposes is the method described in [7], which we call the Ratio+LUT method: first, the ratio z=a/b is calculated with a division operation (to avoid a large LUT storing a two-variable function), second, the one-variable function atan(z) is computed using a LUT. Note that to extend the computation to a full-quadrant atan2, 4 additions would be needed. As seen in Table 2, when using the Ra-

[TABLE 2] Computational cost comparison of our proposed algorithm with various previously-published arctangent algorithms.

Method	/	$*$	+	LUT size (words)	Max. Error (degrees)
[5]-(Eq. 4.13)	1	0	3	-	±4.075
[6]-(Eq. 12)	2	0	4	-	±4.075
Ours	1	0	5	32	±0.249
[2]-(Eq. 2)	1	3	5	-	±0.276
Ratio+LUT	1	0	4	128	±0.224
Ours	1	0	5	64	±0.126
[4]-(Eq. 18)	1	4	4	-	±0.162
[3]-(Eq. 9)	1	3	5	-	±0.086
Ratio+LUT	1	0	4	256	±0.112
Ours	1	0	5	1K	±0.008
[4]-(Eq. 16)	1	7	6	-	±0.008
Ratio+LUT	1	0	4	4K	±0.007

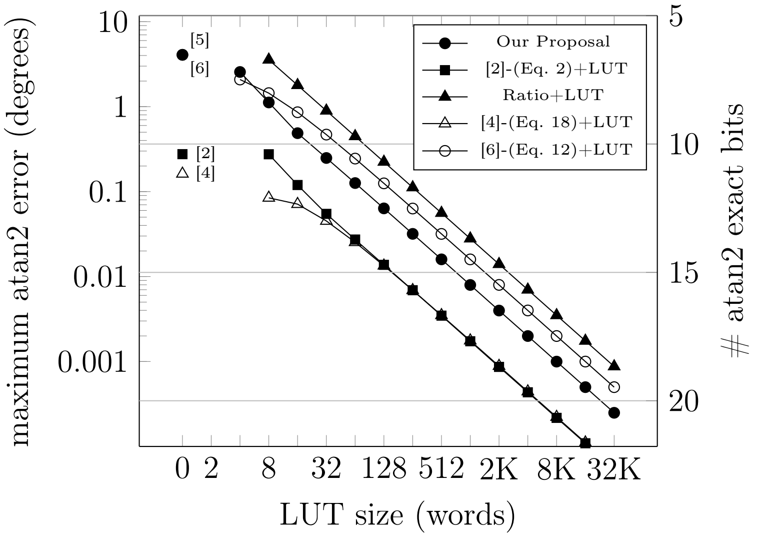
tio+LUT, the size of this LUT would be four times larger than in our proposal, for the same final accuracy. Moreover, in fixed-point implementations of the algorithm, the largest value stored in the atan2 LUT would be 3.5 bits larger than in our case, making the quantization noise worse for the same LUT size and word length.

USING A DIFFERENT FIRST STAGE

As we have shown (see Table 2) the First Stage of our algorithm requires the least arithmetic resources. Nevertheless, in this section we explore the idea of adding a Second Stage based on an error LUT to other atan2 algorithms. Adding a Second Stage could be an easy way of improving the accuracy of an existing implementation with minimum design cost.

We have used [6]-(Eq. 12), [2]-(Eq. 2), and [4]-(Eq. 18) as First Stages in a two-stage algorithm. Their error was computed, using a modified version of the 'errorLUTcontents' function, and the maximum atan2 error was measured for several LUT sizes. The results are shown in Figure 5. When a Second Stage LUT is used, the accuracy is improved significantly. As can be seen, the maximum error is halved (*i.e.* one exact bit more is achieved) when the size of the LUT is doubled.

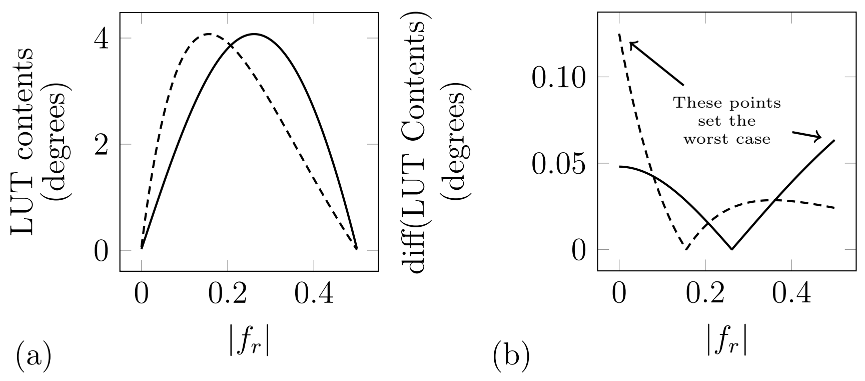
Another interesting observation is that when the Second Stage is added, the atan2 maximum error depends on the maximum first derivative of the error curve stored in the LUT. This explains why, for example, for the same atan2 accuracy, [6]-(Eq. 12) would require a LUT twice as large as ours. Figure 6(a) shows the LUT contents for both approaches: ours with a solid line, [6]-(Eq. 12)'s with a dashed line. Figure 6(b) shows the absolute value of the difference between consecutive values in the LUT, for the specific case of a LUT with 256 words. As can be seen in those figures, our approach has a smaller maximum of the absolute value of the first derivative of the error curve. On the contrary, [4]-(Eq. 18) achieves better accuracy than [2]-(Eq. 2), but when a Second Stage LUT is added their accuracy is similar. That's because in this case they have similar maximum values of the first derivative of their error curves. An important lesson we learn is that atan2 approximations with smaller first derivative error values are better suited for the addition of a Second Stage LUT.



[FIG5] Maximum arctan error and number of exact bits versus LUT size.

CONCLUSIONS

We propose a full-quadrant algorithm for the computation of the arctangent of a complex number c=b+ja, particularly suitable for implementations in hardware, *e.g.* FPGA, ASIC, etc., where there is no penalty incurred when accessing a LUT. The Second Stage of the method we propose could be applied to other low-complexity algorithms for the approximation of the atan2 function, but for a given accuracy there is a trade-off between the complexity of the approximation used for the First Stage and the required storage resources used in the Second Stage. As we have shown, algorithms with a smaller first derivative of



[FIG6] LUT characteristics: a) LUT contents; b) absolute value for the derivatives of the LUT contents for our proposal (solid line) and [6]-(Eq. 12) (dashed line).

their error curve are best suited for improving the accuracy by the addition of a Second Stage LUT. Because our proposed method can be easily improved by increasing the size of a memory when higher accuracy is needed, it is an attractive arctan method in high-speed applications where moderate accuracy is required (*e.g.*, in systems where the precision of the measured a and b variables is, say, 14 bits or less).

REFERENCES

[1] J.-M. Muller, *Elementary functions: algorithms and implementation*. Cambridge, MA, USA; Berlin, Germany; Basel, Switzerland: Birkhäuser, 1997.

[2] R. G. Lyons, "Another contender in the arctangent race," *IEEE Signal Processing Magazine*, vol. 21, no. 1, pp. 109–110, January 2004.

[3] S. Rajan, S. Wang, R. Inkol, and A. Joyal, "Efficient approximations for the arctangent function," *IEEE Signal Processing Magazine*, vol. 23, no. 3, pp. 108–111, May 2006.

[4] X. Girones, C. Julia, and D. Puig, "Full quadrant approximations for the arctangent function," *IEEE Signal Processing Magazine*, vol. 30, no. 1, pp. 130–135, January 2013.

[5] J. M. Shima, "FM demodulation using a digital radio and digital signal processing," Master's thesis, University of Florida, Gainesville, 1995.

[6] S. Winitzki, *Uniform Approximations for Transcendental Functions*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 780–789.

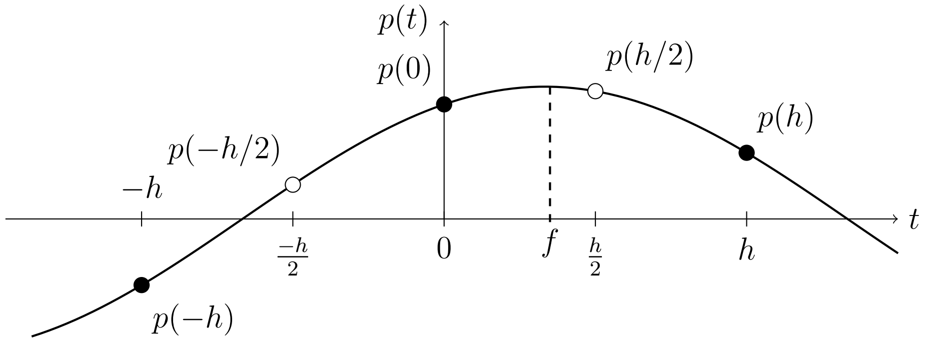
[7] R. Gutierrez and J. Valls, “Implementation on FPGA of a LUT-based atan(y/x) operator suitable for synchronization algorithms.” International Conference on Field Programmable Logic and Applications, 2007, pp. 472–475.

ACKNOWLEDGEMENTS

This work is funded by the Spanish Ministerio de Economía y Competitividad and FEDER under the grant TEC2015-70858-C2-2-R.

APPENDIX

The derivation of (4) proceeds in three steps: (i) derive a polynomial expression approximating Figure A1's continuous $p(t)$ sinusoid in terms of known $p[n]$ samples; (ii) set that expression's time derivative equal to zero; (iii) replace t with f and solve for f.



[FIGA1] Sinusoidal $p(t)$ signal and the desired time value *f*.

Our derivation begins by assuming the largest of our known $p[n]$ samples is located at time t=0. Approximating Figure A1's $p(t)$ sinusoid with a second-order Taylor series expression in the vicinity of t=0, we begin by writing:

$$p(t) \approx p(0) + p'(0)t + \frac{1}{2}p''(0)t^2, \tag{A1}$$

where $p'(0)$ and $p''(0)$ represent the first and second derivatives of $p(t)$ at time $p(t)$=0. Given the (A1) polynomial we next approximate the unknown $p'(0)$ and $p''(0)$ coefficients by using the *central difference formula*. Doing so we write the first-order derivative $p'(0)$ as:

$$p'(0) \approx \frac{p(h) - p(-h)}{2h}. \tag{A2}$$

Having an approximation of $p'(0)$ we next approximate the 2nd-order derivative $p''(0)$ using the first-order derivatives centered at the hypothetical $p(-h/2)$ and $p(h/2)$ samples in Figure A1. Those first-order derivatives are:

$$p'(-h/2) \approx \frac{p(0)-p(-h)}{h} \qquad \textit{and} \qquad p'(h/2) \approx \frac{p(h)-p(0)}{h}.$$

Given $p'(-h/2)$ and $p'(-h/2)$ we write our desired 2nd-order derivative $p''(0)$ as:

$$\begin{aligned} p''(0) &\approx \frac{p'(h/2)-p'(-h/2)}{h} \approx \frac{\frac{p(h)-p(0)}{h} - \frac{p(0)-p(-h)}{h}}{h} \\ &= \frac{p(h)-2p(0)+p(-h)}{h^2}. \end{aligned} \tag{A3}$$

Assuming the time between our known $p[n]$ samples is unity sets h=1 and recalling that for our $p[n]$ samples $p[-1] = -p[1]$, we can rewrite (A2) and (A3) as:

$$p'(0) \approx \frac{p(1)-p(-1)}{2} = p(1) \tag{A4}$$

$$p''(0) \approx \frac{p(1)-2p(0)+p(-1)}{1^2} = -2p(0). \tag{A5}$$

Substituting (A4) and (A5) as coefficients in (A1) our desired approximation of $p(t)$ is:

$$p(t) \approx p(0) + p(1)t - p(0)t^2. \tag{A6}$$

That completes the first step of our derivation. As the second step of our derivation we take the derivative of $p(t)$ to produce:

$$p'(t) \approx p(1) - 2p(0)t. \tag{A7}$$

Setting (A7)'s $p'(t)$=0 gives us an approximation of the time location of the maximum value of Figure A1's $p(t)$ signal. Doing so and defining that estimated time value as f_r we write:

$$0 = p(1) - 2p(0)f_r. \tag{A8}$$

Finally, solving (A8) for our desired expression for f in terms of known $p[n]$ sample values we arrive at the final form of (4) as:

$$f \approx f_r \equiv \frac{-p'(0)}{p''(0)} = \frac{p(1)}{2p(0)}. \tag{A9}$$

AUTHORS

Vicente Torres (vtorres@eln.upv.es) is an associate professor at Universitat Politecnica de Valencia, Valencia, Spain
Javier Valls (jvalls@eln.upv.es) is an associate professor at Universitat Politecnica de Valencia, Valencia, Spain
Richard G. Lyons (r.lyons@ieee.org) is an engineering consultant and lecturer at Besser Associates, Mt. View, California. He is the author of the textbook: "Understanding Digital Signal Processing."