ISSN 2007-9737

Análisis numérico de atan2() para sistemas embebidos

Jacobo Sandoval-Gutiérrez[1], Juan Carlos Herrera-Lozada[2],
Gerardo Abel Laguna-Sánchez[1], Jesús Antonio Álvarez-Cedillo[2]

[1] Universidad Autónoma Metropolitana,
Lerma de Villada,
México

[2] Instituto Politécnico Nacional,
Ciudad de México,
México

{j.sandoval,g.laguna}@correo.ler.uam.mx
{jlozada,jaalvarez}@ipn.mx

Resumen. La función atan2() es utilizada en diferentes áreas del conocimiento, sobre todo mediante el uso de las bibliotecas específicas incluidas con los lenguajes de programación, con las que los usuarios obtienen resultados, más o menos exactos y con cierta precisión, pero sin prestar mayor atención a la cantidad de los recursos computacionales utilizados, a saber, la memoria empleada y la precisión de la unidad lógica-aritmética del procesador. Los usuarios de estas bibliotecas tampoco evalúan otros aspectos relacionados, como lo son el consumo de energía, el espacio utilizado y los costos asociados. Sin embargo, cuando los recursos de cómputo son limitados, como en el caso de los sistemas embebidos, la implementación de toda función matemática requiere de una cierta evaluación de desempeño. En este trabajo se proponen algunas implementaciones para la función atan2(), soportadas por series de Euler y de Maclaurin, realizando la comparación del desempeño obtenido contra las implemntaciones de referencia, a saber, el empleo de tablas de búsqueda y las implementaciones disponibles en las bibliotecas estándares. En este trabajo se aprovecha la arquitectura de los procesadores ARM, haciendo uso sus interrupciones de tipo hilo y sus operaciones vectorizadas, todo ello con el fin de contar con alternativas de implementación para la función atan2(), a fin de poder aplicarlas en dispositivos portátiles y obtener ventajas significativas al lograr un menor tiempo de arranque, menor espacio ocupado, bajo consumo de energía y bajo costo.

Palabras clave. atan2(), ARM, sistemas embebidos.

Numerical analysis of atan2() for embedded systems

Abstract. The atan2() function is used in different areas of knowledge, mainly through the use of specific libraries included with programming languages, with which users obtain results, more or less accurate and with a certain precision, but without paying much attention to the amount of computational resources used, namely the memory used and the precision of the logical-arithmetic unit of the processor. Users of these libraries also do not evaluate other related aspects, such as power consumption, space used, and associated costs. However, when computational resources are limited, as in the case of embedded systems, implementing of any mathematical function requires some performance evaluation. In this work, we propose some implementations for the atan2() function, supported by the Euler and Maclaurin series, comparing the performance obtained against the reference implementations, namely the use of lookup tables and the implementations available in the standard libraries. Furthermore, this work takes advantage of the architecture of ARM processors, making use of their threaded interrupts and vectorized operations, all this to have implementation alternatives for the atan2() function to apply them in portable devices and obtain significant advantages by achieving a shorter startup time, less space occupied, low power consumption and low cost.

Keywords. atan2(), ARM, embedded systems.

ISSN 2007-9737

1. Introducción

Defininción de la función tangente. Dado un punto de origen $O(x_0, y_0)$ y un punto final $P_f(x_f, y_f)$:

$$\tan(\theta) = \frac{y_f - y_0}{x_f - x_0} = \frac{y}{x}; x_f - x_0 \neq 0, \quad (1)$$

En principio, esta función debiera operar en los cuatro cuadrantes, a saber, I: $(+x, +y)$, II: $(-x, +y)$, III: $(-x, -y)$ y IV: $(+x, -y)$. Sin embargo, las implementaciones básicas de la función (1) unicamente consideran los cuadrantes I y III.

Así mismo, para las implementaciones básicas de la función inversa de (1), atan(θ), sólo se considera el intervalo $(-\frac{\pi}{2}, \frac{\pi}{2})$. Es por ello que también se requiere de la función atan2(θ), que sí considera el intervalo completo $(-\pi, \pi]$, es decir los cuatro cuadrantes [1, 18, 26].

El problema de que la implementación computacional básica de la función tangente no considere los cuatro cuadrantes se ha superado mediante de bibliotecas específicas, que se ofrecieron desde las primeras versiones de Fortran [1] hasta los más variados lenguajes como Matlab [16], C++ [6] y PHP [21].

Y, de hecho, no sólo las plataformas como Mozilla [19] y Microsoft [17] han ofrecido bibliotecas con la función atan2(), sino también en los nuevos dispositivos con entornos Android [3] o iOS [13].

Aunque las bibliotecas desarrolladas para arquitecturas de propósito general cuentan con recursos computacionales suficientes, no ocurre lo mismo con otro tipo de dispositivos que carecen de las instrucciones necesarias para realizar operaciones aritméticas complejas en un sólo paso.

Un caso particular de interés son los sistemas embebidos [24], que realizan tareas específicas y donde los recursos computacionales son limitados, en comparación con una computadora de propósito general.

En los sistemas embebidos, toda implementación requiere de un análisis de desempeño, antes de ser aplicada en Hardware, y tiene por objeto optimizar el uso de los recursos para igualar o, incluso, superar el desempeño de una computadora de propósito general.

2. Trabajos relacionados

2.1. Implementaciones en Hardware

Cuando se requiere implementar un algoritmo en una arquitectura específica, partiendo de un código de referencia que se probó originalmente en una arquitectura genérica, existen ciertas restricciones para lograr una implementación que resulte 100 % compatible con la original.

Esto se debe a que cada arquitectura tiene características específicas y, por consiguiente, formas particulares de alcanzar una implementación con desempeño óptimo.

Por ejemplo, en las arquitecturas soportadas por lógica reconfigurable, como en el caso de los FPGA, se han implementado algoritmos CORDIC para el cálculo de las funciones atan() [2, 8, 9, 10, 11, 14] y atan2() [15].

Para el caso de los sistemas embebidos, también existen implementaciones con métodos númerico para las funciones atan() y atan2() [5, 7, 23, 24].

3. Métodos numéricos para calcular atan2()

Algunas formas de representación de la función atan2() son:

– De acuerdo con [27]:

$$\theta = \text{atan2}(a_x, -\sin(\theta_2)a_y + \cos(\theta_2)a_z), \quad (2)$$

donde: $\sin\theta = a_x$ y $\cos\theta = -\sin(\theta_2)a_y + \cos(\theta_2)a_z$.

– De acuerdo con [25]:

$$\theta = \text{atan2}\left(\frac{B_y}{\sqrt{B_y^2 + B_z^2} + B_z}\right). \quad (3)$$

– De acuerdo con [18]:

$$\theta = \text{atan2}\left(\frac{f(x)}{g(x)}\right). \quad (4)$$

ISSN 2007-9737

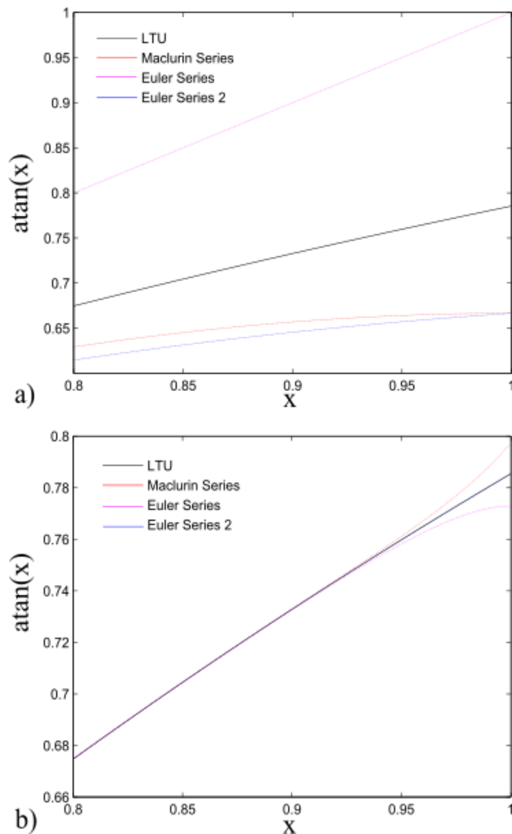

Fig. 1. Resultado de las series para la implementación de atan(), comparadas contra la técnica LTU, para a) una iteración y b) 20 iteraciones

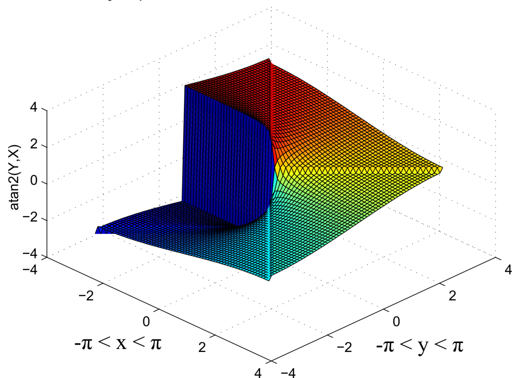

Fig. 2. Resultado de las series para la implementación de $\operatorname{atan} 2()$, con una sola iteración

En particular, en este último caso, se puede aplicar una simple técnica de búsqueda mediante tablas (LUT, por las siglas en inglés de Lookup Table).

Una forma de comprobar la exactitud de la función atan() es evaluar $\sin(4\operatorname{atan}(1))$, que debería corresponder con el cálculo de $\sin(\pi)$.

En principio, el resultado debe ser $\sin(\pi) = 0$ pero, por ejemplo, cuando se realiza esta prueba empleando MATLAB R2021b, se obtiene el siguiente resultado $\sin(\pi) = 1{,}2246 \times 10^{-16}$.

Entonces, no está de más evaluar alternativas de cálculo, para la implementación de la función atan() en cualquier arquitectura, como en las siguientes:

– Mediante la serie de Maclaurin:

$$\operatorname{atan}(x) = \sum_{i=0}^{n} \frac{(-1^i)x^{2i+1}}{(x^{2i+1})}. \tag{5}$$

– Mediante las siguientes series de Euler [24]:

$$\operatorname{atan}(x) = \sum_{i=1}^{n} \frac{(-1^{i+1})}{(2i-1)(x^{2i-1})}, \tag{6}$$

$$\operatorname{atan}(x) = \frac{x}{(1+x^2)} \sum_{j=0}^{n} \prod_{i=1}^{j} \frac{2i * x^2}{(2i+1)(1+x^2)}, \tag{7}$$

donde el término de la suma para $j = 0$ es el producto vacío, al que se le asigna el valor de 1. Para el caso de la función $\sin(x)$, también podemos emplear la siguiente serie:

$$\sin(x) = \sum_{i=0}^{n} \frac{(-1)^i * x^{(1+2i)}}{(1+2i)!}. \tag{8}$$

Entonces, a partir de las series (7) y (8), podemos calcular una aproximación para $\sin(4 * \operatorname{atan}(1))$ y comprobar la exactitud y la precisión de nuestra implementación mediante series.

En particular, para evaluar el desempeño de las series (5), (6) y (7), cada una se probó en el intervalo (0.8,1], con $n = 1$ y $n = 20$, y se comprobó que todas las técnicas convergen cuando $x \ll 1$.

ISSN 2007-9737

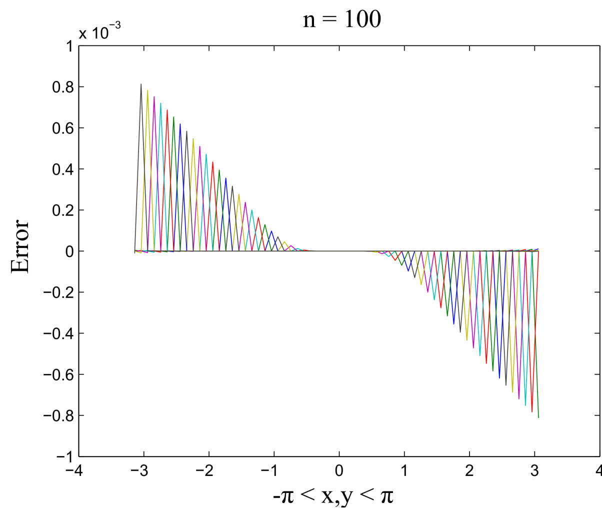

Fig. 3. Error con 100 iteraciones (Orden de magnitud: $e = 1 \times 10^{-3}$)

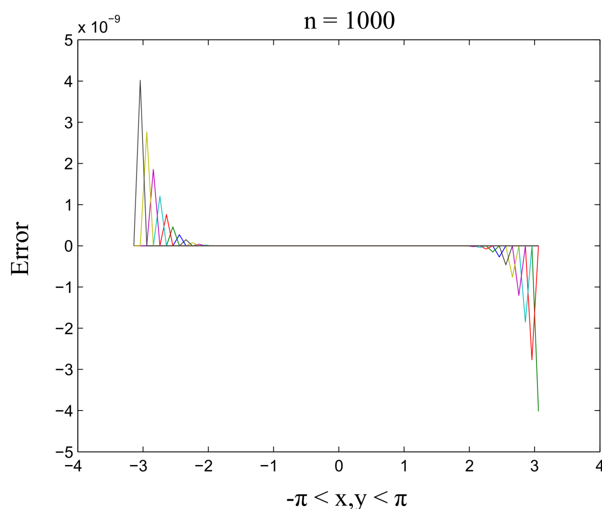

Fig. 4. Error con 1000 iteraciones (Orden de magnitud: $e = 1 \times 10^{-9}$)

También, se observó que la serie (7) converge al valor de mayor precisión, con respecto a las otras dos series, cuando $x = 1$, tal como se muestra en la Fig. 1.

Los resultados anteriores fueron los que motivaron que se optara por la serie (7), como una candidata prometedora, para la implementación de $\operatorname{atan} 2()$ por medio de las siguientes definiciones:

Sea:

$$\theta = \operatorname{atan} 2(y, x), \tag{9}$$

con:

$$\operatorname{atan} 2(y,x) = \begin{cases} \operatorname{atan}\left(\frac{y}{x}\right) & x > 0, \\ \operatorname{atan}\left(\frac{y}{x}\right) + \pi & y \geq 0,\ x < 0, \\ \operatorname{atan}\left(d\frac{y}{x}\right) - \pi & y < 0,\ x < 0, \\ +\frac{\pi}{2} & y > 0,\ x = 0, \\ -\frac{\pi}{2} & y < 0,\ x = 0. \end{cases} \tag{10}$$

Cada una de las pruebas, con la serie en turno para la implementación de (9), se realizó considerando los intervalos $-\pi \leq x \leq \pi$, $-\pi \leq y \leq \pi$ con incrementos de $0{,}1$.

En la Fig. 2, se muestra el resultado para una sola iteración; en la Fig. 3, se presenta el error, con respecto a la función de la biblioteca de referencia, después de 100 iteraciones y, por último, en la Fig. 4, el error después de 1000 interaciones.

Para una iteración de 100 el error se encuentra en un orden de magnitud de 1×10^{-3} y para 1000 iteraciones en un orden de magnitud de 1×10^{-9}, mientras que el error máximo se encuentra en un orden de magnitud de 1×10^{-15}.

4. Arquitectura de los procesadores ARM®

Las familias de los procesadores ARM® se sustentan en una arquitectura con características RISC, es decir, se trata de procesadores con un conjunto de instrucciones reducidas, modo de direccionamiento simple y todas las direcciones destino y fuente determinadas por el registro correspondiente en conjunto con los campos de la instrucción. Las familias más relevantes son las siguientes [4]:

– Cortex-A. Procesadores orientados para la ejecución de sistemas operativos, con aplicaciones en telefonía movil, tabletas, reproductores personales, entre otros.

Tabla 1. Consumo de ciclos de reloj para las operaciones de PF

Tipo de Operación	Ciclos PF a PF	Ciclos Enteros a PF
y[i]=x[i]	8	11
y[i]=x[i]+x[i+1]	16	17
y[i]=x[i]*x[i+1]	16	17
y[i]=x[i]/x[i+1]	16	17

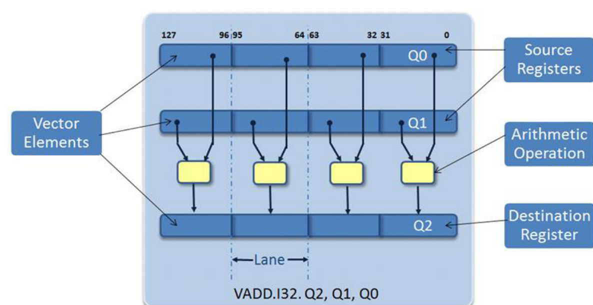

Fig. 5. Unidad NEON Cortex A9 (Cortesía IAR Embedded Workbench)

- Cortex-R. Procesadores para cómputo de tiempo real embebido, sus aplicaciones principales se encuentran en los sistemas de control automotriz y redes de sensores, ya sean conectados o inalámbricos.
- Cortex-M. Procesadores para aplicaciones de cómputo embebido de baja potencia, para estaciones de medición pequeñas, que presuponen el uso de sensores y funciones de microcontrolador.
- Securecore. Procesadores orientados para aplicaciones de máxima seguridad.

5. Dispositivos de procesamiento SITARA y TM4 de Texas Instruments®)

La compañía Texas Instruments, (TI)®, ofrece dispositivos de procesamiento, bajo la denominación de SoC (*Sistem on Chip*), que incluyen procesadores ARM embebidos.

Por ejemplo, la serie SITARA incluye procesadores ARM A9, mientras que los SoC de la serie TM4 incluyen procesadores ARM CORTEX-M4, [22].

La principal ventaja de la unidad de punto flotante vectorizada de un SoC SITARA, soportado por el correspondiente procesador ARM, es que permite realizar operaciones de una sola instrucción para procesar múltiples datos, es decir, incluye instrucciones tipo SIMD, por sus siglas en inglés [12, 20].

La unidad de punto flotante vectorizada, utiliza el estándar IEE754 para la representación con 32 bits, para precisión simple, o con 64 bits, para el caso de precisió doble.

En la representación con 32 bits, el bit 31 es el signo; del bit 30 al 23, el exponente en base 2 y, del bit 22 al 0, la mantisa. En la representación con 64 bits, el bit 63 es el signo; del bit 62 al 52, el exponente y, del bit 51 al 0, la mantisa.

En la Tabla 1, se muestra el consumo de ciclos por operación, obtenidos en forma experimental, al emplear únicamente tipos de punto flotante, así como al realizar la conversión de enteros a punto flotante.

Se observa que se consumen ciclos adicionales cuando se realiza la conversión de entero a flotante. Por lo tanto, aquí se optó por las operaciones que no mezclan pounto flotante con enteros.

Para utilizar operaciones del tipo SIMD, se debe satisfacer que las variables sean indepedientes entre sí, como en el siguiente ejemplo, para el caso de un ciclo *for*:

```
float a[l],b[l];
for (i=0;i<100;i++)
{
    a[i]=a[i+1]=a[i+2]=a[i+3]=0.5;
    b[i]=a[i]+a[i];
}
```

El consumo de ciclos totales para ejecutar la tarea anterior en la forma convencional, con instrucciones para procesar datos aislados, o del tipo SISD, por sus siglas en inglés, se contabilizó en 4608 ciclos; mientras que utilizando la forma vectorizada SIMD se redujo a unicamente 681

ISSN 2007-9737

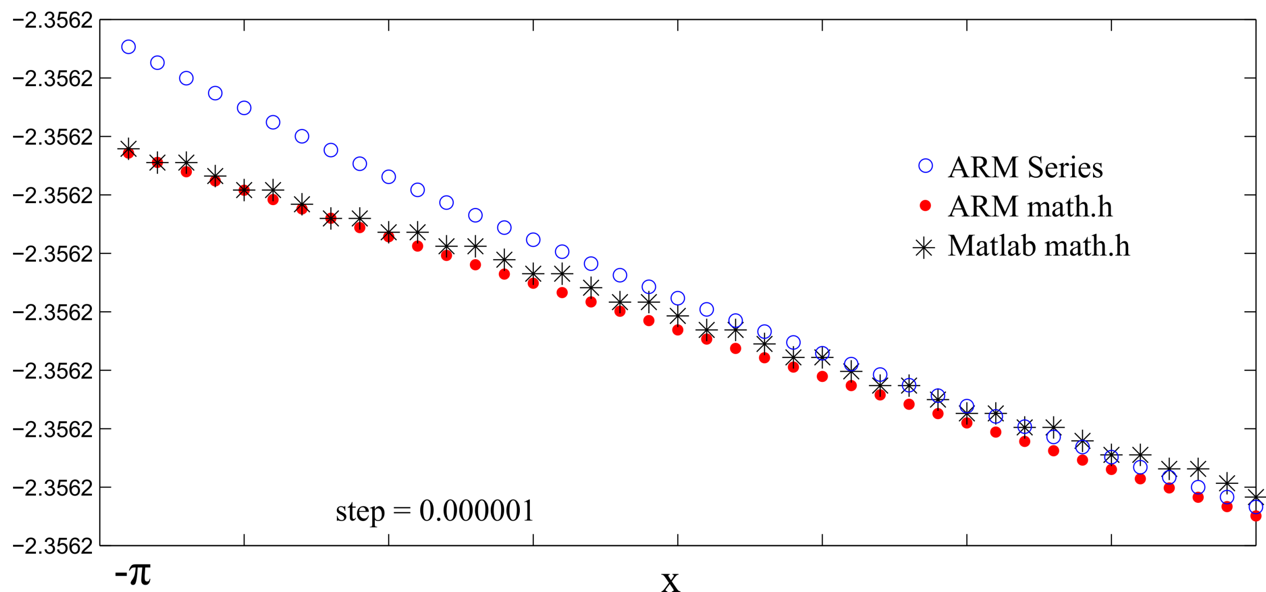

Fig. 6. Comparación de la exactitud obtenida con la implementación de atan2() mediante series, con respecto a la biblioteca math.h

ciclos, una diferencia muy significativa. En la Fig. 5, se muestra cómo es que las operaciones en los elementos del vector a, b y c se realizan en forma simultánea cuando se utilizan las instrucciones de SIMD con el siguiente formato:

```
float a[l],b[l];
#pragma vectorize
for(...
```

En el caso de los SoC TM4, también se cuenta con una unidad de punto flotante vectorizada, sin embargo, sólo se dispone de una precisión 16 bits.

6. Implementación de atan2() para arquitecturas ARM®

Para la implementación de la función atan2(), expresada en (10), empleando la arquitectura ARM®, fue necesario realizar algunos ajustes en la codificación para aprovechar las ventajas de las instrucciones del tipo SIMD.

Para el caso de la expresión (8), es innecesario calcular -1^i, basta con considerar que el resultado es la unidad con un sino que depende de la naturaleza par o impar del exponente.

En el siguiente código se muestra la sección que detremina el valor de -1^i mediante una operación de módulo 2:

```
if (x<1 && x>-1)
{
 for (i=0;i<=n;i++)
 {
   if (i%2==0)
   {f=1;}
   else
   {f=-1;}
   y=y+(f*pow(x,(2*i)+1))/((2*i)+1);
 }
 yt[j][k]=y;
}
```

La biblioteca **math.h** también incluye la función atan2(), sin embargo su exactitud difiere significativamente, con respecto a la propuesta soportada en series.

En la Fig. 6, donde se frafican los resultados obtenidos para x en el intevalo $[-3{,}141607, -3{,}141577]$ con pasos de 1×10^{-6}, se puede apreciar la diferencia de exactitud entre las diferentes implementaciones.

ISSN 2007-9737

Tabla 2. Resolución de los diferentes métodos

Argumento x	Propuesta c/series	MATLAB math.h	ARM math.h
-3.1415**81**	-2.3537223	-2.35619228	-2.3561959
-3.1415**80**	-2.3537228	-2.35619227	-2.3561961
-3.1415**79**	-2.3537230	-2.35619225	-2.35619**64**
-3.1415**78**	-2.3537232	-2.35619224	-2.35619**64**
-3.1415**77**	-2.3537235	-2.35619222	-2.3561966

En el extremo derecho de la Fig. 6, tambien se puede apreciar un par de resultados, obtenidos con la biblioteca math.h, donde se obtiene el mismo resultado para dos valores diferentes de x, tal como se muestra con más detalle en la Tabla 2, lo que indica que, en ese intervalo, la función de la biblioteca math.h no es sensible a cambios en x con el orden de magnitud de 1×10^{-6}.

De acuerdo a las cinco condiciones contempladas en la definición de (10), tres requieren utilizar series, a saber:

$$\left(\operatorname{atan}\left(\frac{y}{x}\right)\right), \tag{11}$$

$$\left(\operatorname{atan}\left(\frac{y}{x}\right) + \pi\right), \tag{12}$$

$$\left(\operatorname{atan}\left(\frac{y}{x}\right) - \pi\right), \tag{13}$$

donde cada una será controlada por una interrupción externa en forma de hilo. En estos casos, una vez activada la interrruрción, se procede a ejecutar las operaciones en forma vectorial para las expresiones (5, 6, 7).

Aunque la exactitud de la implementación de la función $\operatorname{atan} 2$ soportada por series puede ser más exacta, también es patente que el consumo de tiempo es directamente proporcional al número de iteraciones realizadas y determinadas por la variable n.

Por el contrario, el tiempo consumido por la función $\operatorname{atan} 2$ de la biblioteca math.h es prácticamente constante y se mantiene alrededor de $0{,}000105125$ s. A fin de acotar el tiempo de ejecución, es necesario conocer el tiempo de ejecución máximo permitido, para obtener el resultado con una cierta precisión, a fin de determinar un valor adecuado para n. En la Fig. 7, se muestran los tiempos de ejecución para diferentes valores de n.

El tiempo de ejecución, en función de n, satisface el siguiente modelo lineal:

$$t_{(\operatorname{atan} 2)} = n \times 70 \times 10^{-6} + 64 \times 10^{-6}, \tag{14}$$

donde el tiempo de ejecución, $t_{(\text{atan2})}$, se expresa en segundos y n es el número de iteraciones. Despejando n de (14), tenemos:

$$n = \frac{t_{(\operatorname{atan} 2)} - 64 \times 10^{-6}}{70 \times 10^{-6}}. \tag{15}$$

6.1. Ejemplos

– 1. Suponer que se disponde de un tiempo de 10 ms para obtener el resultado de la operación atan2(), sin importar la exactitud obtenida.

  En este caso, el resultado obtenido mediante las series se alcanza con $n = 143$ iteraciones.

– 2. El tiempo mínimo de la ejecución para atan2() con series se logra cuando se considera una sola iteración, es decir, con $n = 1$.

  En este caso el tiempo de ejecución es de $0{,}00008725$ s pero con una exactitud menor, en comparación con el $0{,}000105125$ s que le toma a la función de la biblioteca math.h con una mejor exactitud.

ISSN 2007-9737

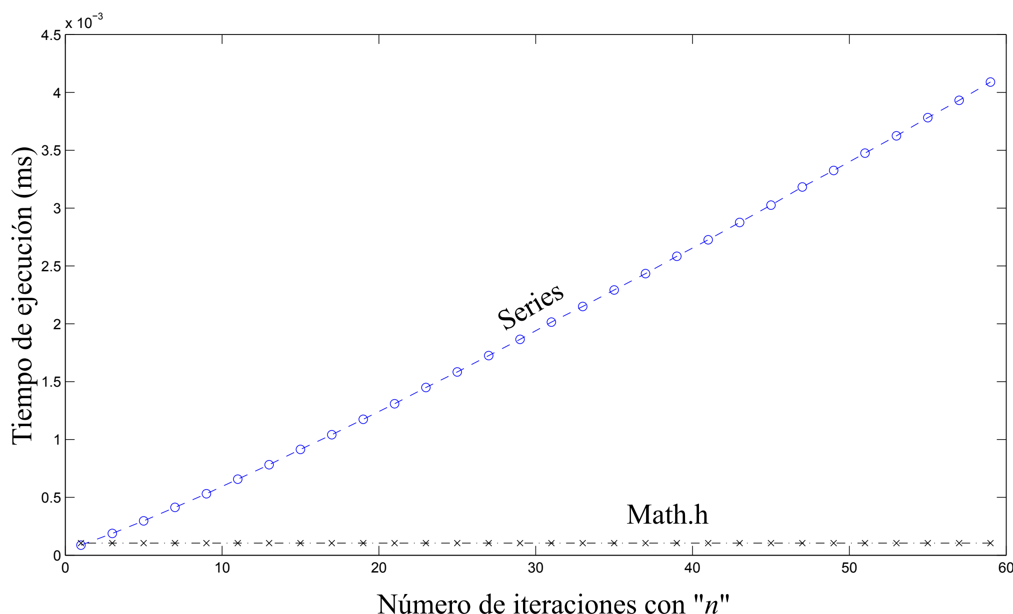

Fig. 7. Tiempo de ejecución para diferentes valores de n

7. Conclusiones

El método propuesto en este trabajo para la función atan2() se soporta por las series de Maclaurin y Euler, obteniendo resultados similares a las funciones incluidas en las bibliotecas estándares en diversos lenguajes de programación.

No obstante lo anterior, la ventaja del método propuesto se hace evidente cuando se dispone de recursos de procesamiento limitados.

Este último caso es precisamente el de los sistemas embebidos, donde la importancia de consumir menos ciclos de reloj, incrementar la resolución y administrar la memoria limitada es de suma importancia.

En este contexto, la propuesta para la función atan2() de este trabajo, satisface el requerimiento de operar de manera eficiente, al reducir alrededor de un 25 % el número de operaciones secuenciales y obteniendo resultados más exactos.

Aunque el tiempo de ejecución de la función atan2() soportada por series es directamente proporcional al número de itearciones realizadas, se pueden alcanzar tiempos de ejecución menores a los de la función de la biblioteca math.h, siempre que se pueda sacrificar cierta exactitud en los cálculos. En este sentido, la propuesta de este trabajo tiene la ventaja de incluir el número de iteraciones de ejecución como un parámetro de operación, que puede ajustarse, según las necesidades del caso, permitiendo obtener resultados razonablemente exactos en un tiempo de ejecución acotado.

Referencias

1. **Agarwal, R. C., Cooley, J. W., Gustavson, F. G., Shearer, J. B., Slishman, G., Tuckerman, B. (1986).** New scalar and vector elementary functions for the ibm system/370. IBM Journal of Research and Development, Vol. 30, No. 2, pp. 126–144. DOI: 10.1147/rd.302.0126.
2. **Andraka, R. (1998).** A survey of CORDIC algorithms for fpga based computers. Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays, pp. 191–200. DOI: 10.1145/275107.275139.
3. **Android (2022).** Strictmath — android developers.
4. **ARM (2022).** C251 user's guide: atan2 library routine.

ISSN 2007-9737

5. **Benammar, M., Alassi, A., Gastli, A., Ben-Brahim, L., Touati, F. (2019).** New fast arctangent approximation algorithm for generic real-time embedded applications. Sensors, Vol. 19, No. 23. DOI: 10.3390/s19235148.

6. **C++ (2022).** std::atan2, std::atan2f, std::atan2l - cppreference.com.

7. **Fons, F., Fons, M., Cantó, E., López, M. (2006).** Trigonometric computing embedded in a dynamically reconfigurable CORDIC system-on-chip. Reconfigurable Computing: Architectures and Applications, pp. 122–127.

8. **Guntoro, A., Zipf, P., Soffke, O., Klingbeil, H., Kumm, M., Glesner, M. (2006).** Implementation of realtime and highspeed phase detector on fpga. Reconfigurable Computing: Architectures and Applications, pp. 1–11. DOI: 10.1007/11802839_1.

9. **Gutierrez, R., Torres, V., Valls, J. (2010).** Fpga-implementation of atan(y/x) based on logarithmic transformation and lut-based techniques. Journal of Systems Architecture, Vol. 56, No. 11, pp. 588–596. DOI: 10.1016/j.sysarc.2010.07.013.

10. **Gutierrez, R., Valls, J. (2007).** Implementation on fpga of a lut-based atan(y/x) operator suitable for synchronization algorithms. 2007 International Conference on Field Programmable Logic and Applications, pp. 472–475. DOI: 10.1109/FPL.2007.4380692.

11. **Gutierrez, R., Valls, J. (2009).** Low-power fpga-implementation of atan(y/x) using look-up table methods for communication applications. J. Signal Process. Syst., Vol. 56, No. 1, pp. 25–33. DOI: 10.1007/s11265-008-0253-z.

12. **Hüsken, M. (2022).** IeeeCC754++ - an advanced set of tools to check IEEE 754-2008 conformity.

13. **iOS (2022).** Mac os x manual page for atan2(3).

14. **Kandeepan, S., Hashmi, O., Zheng, Q. (2007).** A complex-envelope based digital phase locked loop with an arctan phase detector implemented on fpga and performance analysis. Proceedings of the 6th International Conference on Information, Communications and Signal Processing, pp. 1–5. DOI: 10.1109/ICICS.2007.4449626.

15. **Kung, Y.-S., Chen, C.-S. (2008).** FPGA-realization of a motion control IC for robot manipulator. In Robot Manipulators, chapter 16. DOI: 10.5772/6211.

16. **Matlab (2022).** Four-quadrant inverse tangent - matlab atan2.

17. **Microsoft (2022).** Mathf.atan2(single, single) method (system) — microsoft docs.

18. **Moritsugu, S., Matsumoto, M. (1989).** A note on the numerical evaluation of arctangent function. SIGSAM Bull., Vol. 23, No. 3, pp. 8–12. DOI: 10.1145/74646.74647.

19. **Mozilla (2022).** Math.atan2() - javascript — mdn.

20. **Patton, S. (2022).** From start to finish: A product development roadmap for sitara™ processors. .

21. **Php (2022).** Arc tangent of two variables.

22. **TM4 (2013).** Tiva c series tm4c123g launchpad evaluation board users guide. .

23. **Torres, V., Valls, J., Lyons, R. (2017).** Fast- and low-complexity atan2(a,b) approximation [tips and tricks]. IEEE Signal Processing Magazine, Vol. 34, No. 6, pp. 164–169. DOI: 10.1109/MSP.2017.2730898.

24. **Ukil, A., Shah, V. H., Deck, B. (2011).** Fast computation of arctangent functions for embedded applications: A comparative analysis. Proceedings of the IEEE International Symposium on Industrial Electronics, pp. 1206–1211. DOI: 10.1109/ISIE.2011.5984330.

25. **Watanabe, M., Sofko, G. J. (2009).** Role of interchange reconnection in convection at small interplanetary magnetic field clock angles and in transpolar arc motion. Journal of Geophysical Research: Space Physics, Vol. 114, No. 1. DOI: 10.1029/2008JA013426.

26. **Yoshikawa, T. (2003).** Appendix 1 function atan2. In Foundations of Robotics: Analysis and Control. The MIT Press. DOI: 10.7551/mitpress/3074.003.0009.

27. **Zein-Sabatto, S., Bodruzzaman, M., Kuschewski, J. (1995).** Biologically motivated servomechanism kinematic models. Proceedings of the IEEE Southeastcon '95. Visualize the Future, pp. 165–168. DOI: 10.1109/SECON.1995.513078.

Article received on 11/02/2022; accepted on 03/11/2022.
Corresponding author is Juan Carlos Herrera-Lozada.