UNIVERSITAT DE BARCELONA

Trabajo final de Grado

Grado de Matemáticas

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

Las matemáticas de libquadmath

Autor: Brayan Azcarate Montoya

Director:	Dr. Jaume Timoneda Salat
Realizado en:	Departamento de Matemáticas e Informática

Barcelona, 29 de junio de 2017

Abstract

Libquadmath is a library from the GNU GCC Project. It implements mathematical functions in C for 128 bits floating point values with a quad precision according to IEEE 754-2008 standard. The computing of these functions is based on functional approximations and numerical methods such as: Taylor series, Newton-Raphson method and, mainly, in Chebyshev polynomials expansion and minimax approximation. This paper analyses the implemented methods used for an efficient and fast computing of quad precision for square root, cube root, trigonometric, exponential and hyperbolic functions.

Resumen

Libquadmath es una biblioteca del proyecto GNU GCC. Implementa funciones matemàticas en C, para valores de punto flotante de 128 bits con una precisión cuádruple, siguiendo el estandar IEEE 754-2008. El cálculo de estas funciones se basa en aproximaciones funcionales y métodos numéricos tales como: series de Taylor, método de Newton-Raphson y, principalmente, aproximaciones por polinomios de Chebyshev y aproximaciones minimax. En este trabajo se analizan los métodos matemáticos implementados, para conseguir una precisión cuádruple de una forma computacionalmente óptima como en el caso de las raíces cuadrada y cúbica, las funciones trigonométricas, exponencial e hiperbólicas.

Agradecimientos

En primer lugar quiero agradecer al Dr. Jaume Timoneda Salat, tutor de este trabajo, por toda la ayuda que me ha proporcionado y todas las horas que me ha dedicado. También quiero agradecer a mi familia y amigos que me han ayudado y apoyado durante estos meses.

Índice

1. Introducción

La precisión cuádruple surge de la necesidad de obtener cálculos matemáticos con gran precisión.

Así por ejemplo, en el campo de los sistemas dinámicos, *splitting* de variedades invariantes. En ciertas circunstancias, cuando se estudia la dinámica de campos vectoriales o difeomorfismos que dependen de un parámetro pequeño ε, las variedades invariantes de puntos fijos se pueden cortar formando un ángulo del orden de $e^{\frac{K}{\varepsilon^p}}$ con $p > 0$ y hace falta realizar los cálculos usando cifras suficientes. Ya que por ejemplo, si $\varepsilon = 0{,}1$ y $p = 0{,}2$ el ángulo es aproximadamente e^{-100}.

Otro campo donde la precisión cuádruple es necesaria, es el de la mecánica celeste en la estabilidad del sistema solar. Cuando se estudia el movimiento de los objetos del sistema solar, para determinar si quedarán o no confinados en regiones cercanas a la posición actual relativa, se hace la integración numérica durante períodos de tiempo de miles o millones de años. Puesto que la cantidad de operaciones involucradas es muy elevada, y cada operación conlleva un error, hace falta trabajar con muchas cifras para tener unas cuantas cifras significativas en el resultado.

Muchos microprocesadores realizan por *hardware* las operaciones aritméticas elementales (suma, resta, multiplicación y división) y funciones matemáticas en precisiones simple y doble, pero dadas las necesidades es necesario agregar las operaciones y funciones en precisión cuádruple por *software*. La biblioteca *libquadmath* implementa estas funciones más elaboradas, con una precisión cuádruple en el cálculo para valores de punto flotante. La representación en punto flotante $(1.d_1 \cdots d_{p-1} * b^e)$ es una forma de notación que nos permite trabajar con valores de diferente orden de magnitud: por ejemplo, la distancia entre galaxias (magnitud muy grande) o el diámetro de un núcleo atómico (magnitud muy pequeña) y representar estos valores con el mismo número de bits.

El principal objetivo de este trabajo es analizar los diversos métodos implementados para conseguir una precisión cuádruple en el cálculo de diferentes funciones matemáticas. El trabajo se divide en 3 partes claramente diferenciables: una primera parte en la cual se explican los formatos de punto flotante del estándar *IEEE* 754-2008. Una segunda parte donde se estudia la matemática necesaria para la implementación de las funciones. Por último, se explica el desarrollo e implementación de las funciones: raíz cuadrada, raíz cúbica, trigonométricas, exponencial e hiperbólicas. En algunas de estas funciones se aplica el método de *Newton-Raphson* hasta obtener la precisión deseada y en otras funciones la precisión cuádruple se obtiene haciendo una reducción del argumento a un intervalo cercano a cero y aproximando la función por polinomios o cociente de éstos.

2. Estándar IEEE 754-2008

2.1. Introducción

Dada la necesidad de abordar los problemas en las diversas implementaciones de punto flotante, surgió el estándar *IEEE 754* el cual especifica la aritmética de punto flotante en base 2 y base 10. Este estándar fue establecido en 1985, por el *Institute of Electrical and Electronics Engineers (IEEE)*, y actualizado en el 2008 (*IEEE 754-2008*)[1].

De esta forma se logró estandarizar la representación de los números de punto flotante. *IEEE 754-2008* establece las reglas de redondeo, las operaciones aritméticas y las excepciones, tales como la división entre cero, el desbordamiento/subdesbordamiento entre otras. A día de hoy, este estándar está implementado en hardware en la mayoría de microprocesadores.

2.2. Formatos

Cada formato del *IEEE* es usado para representar un subconjunto finito de $\mathbb{R}$ y cada uno está caracterizado por la base, la precisión y el rango del exponente.

Dada una base b, el número de punto flotante se describe por 3 parámetros: el signo, el exponente sin sesgo y la mantisa.

$$(-1)^{signo} \cdot b^{exponente} \cdot mantisa$$

Los formatos especificados en el *IEEE* son:

- *Single* con una longitud de 32 bits.
- *Double* con una longitud de 64 bits.
- *Quadruple* con una longitud de 128 bits.

Y una aritmética especial para el formato extendido con una longitud de 80 bits, que corresponde al formato extendido de la FPU de INTEL.

Codificación y números representables

Una codificación asigna una representación de un dato de punto flotante a una cadena de bits. El conjunto finito de números de punto flotante representable dentro de un formato determinado está definido por los siguientes parámetros enteros:

- b= la base, puede ser 2 o 10.

- p= Número de dígitos significativos, es decir, la precisión.
- e_{max}= Exponente máximo.
- e_{min}= Exponente mínimo ($e_{min} = 1 - e_{max}$).

Parámetro	***Single***	***Double***	***Quadruple***
Nº de bits de almacenamiento: k	32	64	128
Precisión: $p = k - 4 \cdot log_2(k) + 13$	24	53	113
$e_{max} = 2^{k-p-1} - 1$	127	1023	16383
$e_{min} = 1 - e_{max}$	-126	-1022	-16382

Tabla 1: Parámetros formatos binario

Formato	k	p	e_{max}	e_{min}
Long-Double	80	65	16383	-16382

Tabla 2: Parámetros *Double-extendido*

Dentro de cada formato, se representarán los siguientes datos de punto flotante:

- ± 0
- Números de punto flotante diferentes de cero, de la forma $(-1)^s \cdot b^e \cdot m$. Donde
  - s es 0 ó 1.
  - $e \in \mathbb{Z}$ tal que $e_{min} \leq e \leq e_{max}$
  - m es un número representado por una cadena de dígitos de la forma $1.d_1...d_{p-1}$ donde d_i es un entero tal que $0 \leq d_i < b$
- Dos infinitos ($\pm\infty$)
- Dos tipos de NaN *(Not a Number)*: Un NaN silencioso (qNaN) y un NaN de señalización (sNaN), este último sirve para señalar una condición de operación no válida.

El número de punto flotante *normal* más pequeño es $b^{e_{min}}$ y el más grande es $b^{e_{min}} \cdot (b - b^{1-p})$. Los números de punto flotante distintos de cero con una magnitud menor que $b^{e_{min}}$ se llaman *desnormalizados* porque su valor absoluto se encuentra entre cero y el *normal* más pequeño.

2.3. Formato de codificación en binario

Cada número de punto flotante tiene una única codificación en formato binario. Para hacer la representación $(-1)^s \cdot b^e \cdot m$ única, el valor de la mantisa m se maximiza dividiendo e entre 2, hasta que $e = e_{min}$ o $m \geq 1$. Este proceso se llama *normalización*. Si después del proceso $e = e_{min}$ y $0 < m < 1$, el número de punto flotante es *desnormalizado*. Los números *desnormalizados* (y cero) se codifican con un valor de exponente sesgado reservado para ellos.

El proceso de *normalización* conduce a que un número *normal* distinto de cero se exprese de la forma $2^e\,(1 + f)$ con $0 < f < 1$.

las representaciones de punto flotante en binario son codificadas en K bits de la siguiente forma:

- 1 bit del signo
- w bits del exponente sesgado. $E = e + sesgo$
- t bits significativos de la mantisa. Donde $t = p - 1$

Parámetro	***Single***	***Double***	***Long-Double***	***Quadruple***
$sesgo = e_{max}$	127	1023	16383	16383
Bit signo	1	1	1	1
Bits exponente sesgado w	8	11	15	15
Bits mantisa t	23	52	64	112

Tabla 3: Codificación de los parámetros

El rango de codificación del exponente sesgado incluye:

- cualquier $n \in \mathbb{N}$ tal que $1 \leq n \leq 2^w - 2$, para codificar números enteros.
- el valor reservado 0 para codificar ± 0 y números *desnormalizados*.
- el valor reservado $2^w - 1$ para codificar $\pm\infty$ y NaN.

La siguiente tabla muestra la amplitud y precision de cada formato binario de punto flotante:

Formato	**Min. Denormal**	**Min normal**	**Max finito**	**Decimales sig.**
Single	$1{,}4 \cdot 10^{-45}$	$1{,}2 \cdot 10^{-38}$	$3{,}4 \cdot 10^{38}$	7
Double	$4{,}9 \cdot 10^{-324}$	$2{,}2 \cdot 10^{-308}$	$1{,}8 \cdot 10^{308}$	15
Long-Double	$1{,}8 \cdot 10^{-4951}$	$3{,}4 \cdot 10^{-4932}$	$1{,}2 \cdot 10^{4932}$	19
Quadruple	$6{,}5 \cdot 10^{-4966}$	$3{,}4 \cdot 10^{-4932}$	$1{,}2 \cdot 10^{4932}$	32

Tabla 4: Precisión y amplitud [2]

Donde:

- Número denormal mínimo positivo: $\frac{2^{2-2^{14}}}{2^{(p-1)}}$
- Número normal mínimo positivo: $2^{e_{min}}$
- Número máximo positivo: $\left(1-\frac{1}{2^p}\right)\cdot 2^{2^{(w-1)}}$
- Decimales significativos $\lfloor log_{10}(2)\cdot p\rfloor$

Donde $\lfloor x\rfloor$ denota la función suelo. $\lfloor x\rfloor := \text{mín}\{k\in\mathbb{Z}|k\leq x\}$

3. Métodos numéricos

3.1. Propagación de errores

La propagación de errores es el efecto del error de las variables en el error de una función, normalmente este error se define como el error absoluto. En las operaciones en punto flotante, aunque los valores sean exactos, el resultado puede no ser exacto por efecto del redondeo. Recordemos que los errores en las magnitudes se transmiten a los resultados cuando hacemos una operación aritmética. Las fórmulas de propagación de los errores las vemos a continuación.

Propagación de errores en sumas y diferencias

Sea $x=\bar{x}\pm\varepsilon x$, $y=\bar{y}\pm\varepsilon y$. Si $z=x\pm y$, entonces

$$z=\bar{x}\pm\bar{y}+\varepsilon x+\varepsilon y\Longrightarrow\boxed{\varepsilon z\approx\varepsilon x+\varepsilon y}$$

El error absoluto de la suma y la diferencia de dos o mas valores es la suma de los errores absolutos de dichos valores.

Propagación de errores en productos

Sea $x=\bar{x}\pm\varepsilon x=\bar{x}\left(1\pm\frac{\varepsilon x}{|\bar{x}|}\right)$, $y=\bar{y}\pm\varepsilon y=y\left(1\pm\frac{\varepsilon y}{|\bar{y}|}\right)$. Si $z=x\cdot y$, entonces

$$z=\bar{x}\left(1\pm\frac{\varepsilon x}{|\bar{x}|}\right)\cdot\bar{y}\left(1\pm\frac{\varepsilon y}{|\bar{y}|}\right)\approx\bar{x}\bar{y}\cdot\left(1\pm\left(\frac{\varepsilon x}{|\bar{x}|}+\frac{\varepsilon y}{|\bar{y}|}\right)\right)\qquad\left(\frac{\varepsilon x\cdot\varepsilon y}{|\bar{x}|\,|\bar{y}|}\ll 1\right)$$

$$\Longrightarrow\frac{\varepsilon z}{|\bar{z}|}\approx\frac{\varepsilon x}{|\bar{x}|}+\frac{\varepsilon y}{|\bar{y}|}$$

El error relativo de z es la suma de errores relativos.

Propagación de errores en cocientes

Sea $x = \bar{x} \pm \varepsilon x = \bar{x}\left(1 \pm \dfrac{\varepsilon x}{|\bar{x}|}\right)$, $y = \bar{y} \pm \varepsilon y = \bar{y}\left(1 \pm \dfrac{\varepsilon y}{|\bar{y}|}\right)$. Si $z = \dfrac{x}{y}$, entonces

$$z = \frac{\bar{x}\left(1 \pm \frac{\varepsilon x}{|\bar{x}|}\right)}{\bar{y}\left(1 \pm \frac{\varepsilon y}{|\bar{y}|}\right)} \approx \frac{\bar{x}}{\bar{y}} \cdot \left(1 \pm \left(\frac{\varepsilon x}{|\bar{x}|} + \frac{\varepsilon y}{|\bar{y}|}\right)\right)$$

$$\Longrightarrow \frac{\varepsilon z}{|\bar{z}|} \approx \frac{\varepsilon x}{|\bar{x}|} + \frac{\varepsilon y}{|\bar{y}|}$$

El error relativo de z es la suma de errores relativos.

Propagación de errores en una función

Sea $f(x)$ una función cualquiera. Sea $x = \bar{x} \pm \varepsilon x$. Si x se utiliza para calcular $z = f(x)$, la propagación del error está determinado por

$$\varepsilon z = f(\bar{x} \pm \varepsilon x) - f(\bar{x}) \Rightarrow \varepsilon z \approx \left|\frac{df(\bar{x})}{d\bar{x}}\right| \varepsilon x$$

Fórmula general de la propagación del error

Sea $z = f(x_1, x_2, \cdots, x_n)$. El error absoluto de la estimación de z es producido por los errores en las variable x_i. Éste viene determinado por

$$\boxed{\varepsilon z \approx \sum_{i=1}^{n} \left|\frac{\partial f}{\partial x_i}\right| \varepsilon x_i} \qquad [3]$$

3.2. Método de Newton-Raphson

El método de *Newton-Raphson* es un algoritmo para resolver la ecuación $f(x) = 0$.

Sea $f : [a, b] \to \mathbb{R}$ una función de clase C^1 en $[a, b]$ ($f \in C^1[a, b]$). La idea básica del método de *Newton-Raphson* es aproximar la función f por su tangente l en una aproximación de la raíz α y resolver la ecuación $l(x) = 0$. Tomamos esta solución como una nueva aproximación de la raíz de $f(x)$ y repetimos el proceso hasta obtener la raíz con la precisión buscada.

Sea x_0 una aproximación inicial de la solución, consideramos la recta tangente a $f(x)$ en $(x_0, f(x_0))$ y tomamos x_1 como la intersección entre la recta tangente y el eje de las abscisas.

La ecuación de la recta tangente es

$$y = f(x_0) + f'(x_0)(x - x_0) \tag{3.1}$$

y la intersección con el eje de las abscisas se obtiene haciendo $y = 0$

$$x_1 \equiv x = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Procediendo de forma iterativa, obtenemos

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \qquad n = 0, 1, 2... \tag{3.2}$$

Siempre y cuando f' no se anule en x_n.

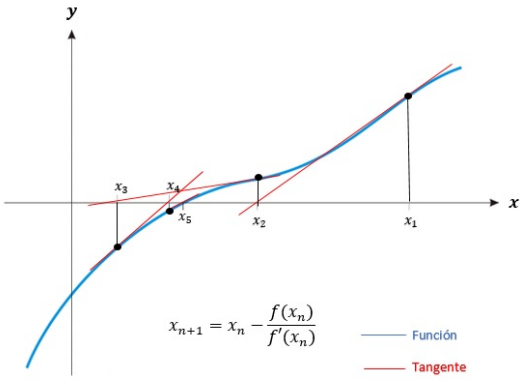

Figura 1: Ejemplo método de *Newton-Raphson*

El método de *Newton-Raphson* converge a la raíz α en condiciones adecuadas. Las condiciones de parada del método pueden ser

$$|f(x_n)| \leq \varepsilon, \qquad |x_{n+1} - x_n| \leq \varepsilon \qquad o \qquad \frac{|x_{n+1} - x_n|}{|x_n|} \leq \varepsilon \qquad [4].$$

Definición 3.1. *Sea $\{x\}_{n=0}^{\infty}$ una sucesión que converge a α, y sea $\varepsilon_n = x_n - \alpha$. Si existe un número p y una constante $C \neq 0$ tal que*

$$\lim_{n\to\infty} \frac{|\varepsilon_{n+1}|}{|\varepsilon_n|^p} = C$$

Entonces p se llama el orden de convergencia de la sucesión y C es la constante de error asintótica.

3.3. Convergencia del método de Newton-Raphson

Supongamos que α raíz simple de $f(x) = 0$ y que $f \in C^2[a, b]$. Entonces tenemos que $f'(\alpha) \neq 0$, $f''(\alpha) \neq 0$ y por tanto $f'(x) \neq 0$ en un entorno cercano a la raíz α. Sea ε_n el error de la estimación x_n, es decir, $\varepsilon_n = x_n - \alpha$.

Expandimos f en una serie de Taylor alrededor de x_n, tenemos

$$0 = f(\alpha) = f(x_n) + f'(x_n)(\alpha - x_n) + \frac{1}{2}f''(\xi)(\alpha - x_n)^2$$

Donde ξ está entre x_n y α. Como $f'(x_n) \neq 0$ podemos dividir por $f'(x_n) \neq 0$

$$\frac{f(x_n)}{f'(x_n)} + \alpha - x_n = -\frac{f''(\xi)}{2f'(x_n)}(\alpha - x_n)^2$$

por la ecuación(3.2), tenemos

$$\alpha - x_{n+1} = -\frac{f''(\xi)}{2f'(x_n)}(\alpha - x_n)^2$$

Por tanto

$$\varepsilon_{n+1} = \frac{1}{2}\varepsilon_n^2 \frac{f''(\xi)}{f'(x_n)}$$

$$\frac{\varepsilon_{n+1}}{\varepsilon_n^2} = \frac{1}{2}\frac{f''(\xi)}{f'(x_n)}$$

Y como $x_n \longrightarrow \alpha$

$$\frac{\varepsilon_{n+1}}{\varepsilon_n^2} \longrightarrow \frac{1}{2}\frac{f''(\alpha)}{f'(\alpha)} = C$$

Por tanto tenemos que el método de *Newton-Raphson* tiene convergencia cuadrática [5].

4. Aproximación funcional

Queremos evaluar una cierta función muchas veces. Se sabe a priori que los argumentos de esta función estarán en algún intervalo, pero no se conoce a priori estos argumentos. Por este motivo se debe aproximar la función sobre el intervalo total.

Sobre la aproximación de la función queremos estar capacitados para acotar el error en el resultado. Dado que a priori no conocemos los valores involucrados,

debemos considerar el peor caso posible. Por tanto, lo más importante de la magnitud a controlar es el error máximo en valor absoluto sobre el intervalo. Por consiguiente, nuestro objetivo es hacer el error máximo tan pequeño como sea posible. A una aproximación de tal tipo la llamaremos *minimax*

Teorema 4.1. ***(Weierstrass).*** *Si $f(x)$ es una función continua sobre un intervalo $[a,b]$, entonces dado cualquier $\varepsilon > 0, \exists n \in \mathbb{N}$, que depende de ε $(n(\varepsilon))$, y existe un polinomio $P_n(x)$ de grado n tal que*

$$|f(x) - P_n(x)| < \varepsilon \qquad \forall x \in [a,b]$$

Definición 4.2. *$\varepsilon_a(x) = |f(x) - P_n(x)|$ se denomina error absoluto de la aproximación $P_n(x)$. Mientras que*

$$\varepsilon_r(x) = \frac{|f(x) - P_n(x)|}{|f(x)|}$$

Se denomina error relativo.

4.1. Serie de Taylor

Teorema 4.3. *[6] Sea f una función de clase C^n en $[a,b]$. Supongamos que $f^{(n+1)}$ existe en $[a,b]$. Sea $x_0 \in [a,b]$. Entonces, para cada $x \in [a,b]$, existe un punto $\xi(x)$ entre x_0 y x tal que*

$$f(x) = P_n(x) + R_n(x)$$

Donde

$$P_n(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n$$

$$R_n(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x - x_0)^{n+1}$$

$P_n(x)$ se llama el n-ésimo polinomio de Taylor de f al rededor de x_0. $R_n(x)$ se llama el error de truncamiento asociado al polinomio $P_n(x)$.

- *Cuando $n \longrightarrow \infty$, $P_n(x)$ se denomina serie de Taylor al rededor de x_0. Si $x_0 = 0$, se suele llamar serie de Maclaurin.*
- *El error de truncamiento se refiere al error que se comete al usar una suma finita al aproximar la función.*

4.2. Error minimax

La idea es minimizar la distancia entre $f(x)$ y $P_n(x)$ en norma L_∞, es decir,

$$\min_{P_n} \| f - P_n \|_\infty = \min_{P_n} (\max_{x\in[a,b]} |f(x) - P_n(x)|$$

Definición 4.4. *La aproximación polinomial minimax de grado n de una función continua $f(x)$ sobre un intervalo $[a, b]$, se define como el polinomio P_n^* tal que*

$$\varepsilon^* = \max_{a\le x\le b} |f(x) - P_n^*(x)| \le \max_{a\le x\le b} |f(x) - P_n(x)|$$

Donde $P_n(x)$ es cualquier polinomio de grado n

4.2.1. Aproximaciones de Padé

Queremos generar aproximaciones de la forma

$$R_{m,k} = \frac{P_m(x)}{Q_k(x)} \qquad [7] \tag{4.1}$$

Para una m y una k dadas escogemos $P_m(x)$ y $Q_k(x)$ de tal manera que $f(x)$ y $P_m(x)/Q_k(x)$ sean iguales en $x = 0$ y tengan tantas derivadas iguales en $x = 0$ como sea posible. Llamamos $N = m + k$ al índice de $R_{m,k}$.

En caso de $k = 0$, la aproximación $R_{m,k}$ es el desarrollo de Maclaurin para $f(x)$. Suponemos que $P_m(x)$ y $Q_k(x)$ no tienen factores comunes, entonces consideramos

$$P_m(x) = \sum_{i=0}^{m} a_i x^i$$

$$Q_k(x) = \sum_{i=0}^{k} b_i x^i$$

Podemos considerar $b_0 = 1$, ya que el término constante no puede ser 0 si la aproximación existe en $x = 0$.
Consideramos ahora que $f(x)$ tiene una serie de Maclaurin

$$f(x) = \sum_{i=0}^{\infty} c_i x^i$$

Entonces consideramos

$$f(x) - \frac{P_m(x)}{Q_k(x)} = \frac{\sum_{i=0}^{\infty} c_i x^i \sum_{i=0}^{k} b_i x^i - \sum_{i=0}^{m} a_i x^i}{\sum_{i=0}^{k} b_i x^i} \tag{4.2}$$

Dado que tenemos $N+1$ constantes, $a_0, ..., a_m$ y $b_1, ..., b_k$, haciendo $f(x) - R_{m,k}(x)$ y sus primeras N derivadas iguales a cero en $x = 0$ obtenemos

$$\left(\sum_{i=0}^{\infty} c_i x^i\right)\left(\sum_{i=0}^{k} b_i x^i\right) - \sum_{i=0}^{m} a_i x^i = \sum_{i=N+1}^{\infty} d_i x^i \tag{4.3}$$

La anulación de los coeficientes de las primeras $N+1$ potencias de x en el primer miembro de (4.3) es equivalente a

$$\begin{aligned} \sum_{i=0}^{k} c_{N-s-i} b_i = 0 \qquad & s = 0, 1, ..., N-m-1 \\ & (c_i = 0 \quad si \quad i < 0, b_0 = 1) \\ a_r = \sum_{i=0}^{r} c_{r-i} b_i \qquad & r = 0, 1, ..., m \\ & (b_i = 0 \quad si \quad i > k) \end{aligned}$$

Cuando este sistema tiene solución, ésta nos proporciona la aproximación deseada de la forma (4.1).

Los coeficientes d_i y b_i decrecen muy rápidamente en magnitud. Por consiguiente, una buena estimación del error de la aproximación $R_{m,k}$ puede estar dada por $d_{N+1}x^{N+1}$. Donde

$$d_{N+1} = \sum_{i=0}^{k} c_{N+1-i} b_i$$

4.2.2. Polinomios de Chebyshev

Las aproximaciones de Padé no son las mejores desde el punto de vista *minimax*, sin embargo nos sirven como punto inicial para determinar mejores aproximaciones. El problema de las aproximaciones basadas en series de Maclaurin, es que el error sobre un intervalo centrado en cero no es uniforme, puesto que el error es pequeño alrededor del 0, pero crece rápidamente cuando nos alejamos hacia los extremos. Por tanto, usaremos polinomios cuyo comportamiento en un intervalo centrado en cero sea más uniforme: los polinomios de Chebyshev.

Definición 4.5. *el polinomio de Chebyshev de primera especie de grado n, está definido por:*

$$\begin{aligned} T_n(x) :&= \cos(n \arc\cos x) \\ T_n(\cos\theta) &= \cos(n\theta) \end{aligned}$$

El coeficiente de x^n *es* 2^{n-1}.

Estos polinomios son ortogonales respecto al peso $\dfrac{1}{\sqrt{1-x^2}}$ en el intervalo $[-1,1]$.

$$\int_{-1}^{1} \frac{T_n(x)T_m(x)}{\sqrt{1-x^2}}dx = \begin{cases} 0 & \text{si } n \neq m \\ \pi & \text{si } n = m = 0 \\ \frac{\pi}{2} & \text{si } n = m \neq 0 \end{cases}$$

Además, los polinomios de Chebyshev satisfacen la siguiente relación de recurrencia

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \qquad \text{con} \qquad T_0(x) = 1, \quad T_1(x) = x$$

Podemos suponer, sin pérdida de generalidad, que el intervalo sobre el que queremos aproximar la función es $[-1,1]$. Ya que en caso de tener otro intervalo $[a,b]$ podemos reducirlo a $[-1,1]$ mediante un cambio de variable. Si denominamos $\bar{x}$ a la variable en $[-1,1]$, la transformación es:

$$\bar{x} = -1 + 2\frac{x-a}{b-a} = \frac{2}{b-a}x - \frac{a+b}{b-a}$$

Proposición 4.6. *El polinomio $T_n(x)$ tiene n raíces en el intervalo $[-1,1]$. Las raíces son:*

$$x_k = \frac{\cos(2k+1)\pi}{2n} \qquad k = 0, 1, ..., n-1$$

Además, T_n tiene $n+1$ extremos con valor absoluto igual a 1 y con cambio alternado de signo, que se alcanzan en:

$$\xi_j = \cos\left(\frac{j\pi}{n}\right) \qquad j = 0, 1, ..., n$$

esto es

$$T_n(\xi_j) = (-1)^j$$

Teorema 4.7. ***(Chebyshev)**. De todos los polinomios mónicos de grado n, el polinomio de Chebyshev $\dfrac{1}{2^{n-1}}T_n(x)$ oscila con amplitud mínimo-máxima en el intervalo $[-1,1]$.*

4.2.3. Desarrollo de Chebyshev

El desarrollo de $f(x)$ en una serie de polinomios de Chebyshev está dado por

$$f(x) = \frac{1}{2}c_0 + \sum_{n=1}^{\infty} c_n T_n(x) \tag{4.4}$$

Usando la propiedad de ortogonalidad de los polinomios de Chebyshev, tenemos que

$$c_n = \frac{2}{\pi}\int_{-1}^{1} \frac{f(x)T_n(x)}{\sqrt{1-x^2}}dx$$

Usaremos el desarrollo (4.4) para generar aproximaciones racionales.

Queremos determinar aproximaciones de la forma

$$T_{m,k}(x) = \frac{\sum_{i=0}^{m} a_i T_j(x)}{\sum_{i=0}^{k} b_i T_i(x)} \qquad n = 0, 1, 2, ...$$

Donde las a_i y las b_j son determinadas de tal manera que en

$$f(x) - T_{m,k}(x) = \frac{\left[\frac{1}{2}c_0 + \sum_{i=0}^{\infty} c_i T_i(x)\right]\left[\sum_{i=0}^{k} b_i T_i(x)\right] - \sum_{i=0}^{m} a_i T_i(x)}{\sum_{i=0}^{k} b_i T_i(x)}$$

los coeficientes de $T_i(x)$, $i = 0, ..., N$, en el numerador se anulen. Por tanto

$$\left[\frac{1}{2}c_0 + \sum_{i=0}^{\infty} c_i T_i(x)\right]\left[\sum_{i=0}^{k} b_i T_i(x)\right] - \sum_{i=0}^{m} a_i T_i(x) = \sum_{i=N+1}^{\infty} h_i T_i(x) \qquad (4.5)$$

Usando la identidad

$$T_{i+j}(x) + T_{|i-j|}(x) = 2T_i(x)T_j(x)$$

Tenemos que

$$\frac{1}{2}c_0 + \sum_{i=0}^{\infty} c_i T_i(x) + \frac{1}{2}\sum_{j=1}^{\infty}\sum_{i=0}^{k} b_i c_j \left[T_{i+j}(x) + T_{|i-j|}(x)\right] - \sum_{i=0}^{m} a_i T_i(x) = \sum_{i=N+1}^{\infty} h_i T_i(x)$$

Y obtenemos el siguiente conjunto de ecuaciones

$$a_r = \begin{cases} \frac{1}{2}\sum_{i=0}^{k} b_i c_i & \text{si } r = 0 \\ \frac{1}{2}\sum_{i=0}^{k} b_i \left(c_{|r-i|} + c_{r+i}\right) & \text{si } r = 1, ..., N \\ 0 & \text{si } r > m \end{cases}$$

El primer coeficiente diferente de cero en (4.5) está dado por

$$h_{N+1} = \frac{1}{2}\sum_{i=0}^{k} b_i \left(c_{N+1-i} + c_{N+1+i}\right)$$

Este coeficiente multiplicado por $T_{N+1}(x)$ es frecuentemente una buena aproximación del error en $T_{m,k}(x)$.

Fórmula aproximación de Chebyshev

Sea $f(x)$ una función de clase C^{n+1}, $P_n(x)$ el polinomio de grado n que corresponde al desarrollo de la serie de Chebyshev de $f(x)$ para $x \in [0, \delta]$. Sea $\varepsilon_a = \text{máx}_{0 \leq x \leq \delta} |f(x) - P_n(x)|$, entonces se tiene

$$\boxed{\varepsilon_a < \frac{\delta^{n+1}}{2^n \cdot (n+1)!} \cdot f^{(n+1)}(0)} \qquad [8]$$

5. Aplicaciones del método de Newton-Raphson

Algunas funciones se calculan obteniendo una aproximación inicial con pocos bits significativos y después, aplicando el método de *Newton-Raphson* vamos teniendo más cifras significativas hasta obtener una precisión de 113 bits significativos.

5.1. Raíz cuadrada

Las raíces cuadradas de x son las raíces de la ecuación

$$y^2 - x = 0 \tag{5.1}$$

Si $x > 0$, la ecuación (5.1) tiene 2 raíces reales de igual magnitud y signo opuesto. La función *sqrtq* calcula el valor de la raíz positiva, con una precisión de 113 bits.

Dado un argumento x, primero comprobamos que no se trata de un valor *NaN* o *INF* (infinito), de ser así generamos un error. Si $x < 0$ retornamos un NaN

mediante la expresión $\frac{x-x}{x-x}$, esto nos produce un mensaje de error.

Si el número al cual queremos calcular la raíz cuadrada admite una representación como un *double*, es decir, $DBL_{Min} \leq x \leq DBL_{Max}$, calculamos la raíz cuadrada de x con la función $sqrt(x)$. Esta función calcula la raíz cuadrada de un valor *double* con una precisión de 53 bits significativos.

$$y_0 = sqrt(x)$$

A fin de obtener una precisión una precisión cuádruple, hacemos 2 iteraciones del método de *Newton-Raphson* para la función $f(y) = y^2 - x$ con punto inicial y_0.

$$y_1 = y_0 - \frac{y_0^2 - x}{2y_0} \tag{5.2}$$

Simplificamos (5.2) y obtenemos una expresión equivalente

$$y_1 = y_0 - 0{,}5 \cdot \left(y_0 - \frac{x}{y_0} \right) \tag{5.3}$$

En las 2 expresiones la propagación del error, sin tener en cuenta el error en las operaciones, es igual ya que la derivada respecto de y_0 es igual en ambas expresiones. Pero, (5.3) tiene menor número de operaciones y cada operación conlleva un error. Por lo tanto, esta expresión tiene menor error de propagación y en consecuencia es mejor. Entonces

$$y_2 = y_1 - 0{,}5 \cdot \left(y_1 - \frac{x}{y_1} \right)$$

Cada iteración del método requiere: dos sumas, una multiplicación y una división. Tomando $y_n = (1 + \varepsilon_n)\sqrt{x}$ y remplazando en la expresión (5.3), encontramos que

$$\varepsilon_{n+1} = \frac{1}{2} \frac{\varepsilon_n^2}{1 + \varepsilon_n}$$

Es decir, dado que el método de *Newton-Raphson* tiene una convergencia cuadrática, si el error relativo en la n-ésima iteración es ε, entonces el error en la (n+1)-ésima iteración es aproximadamente $\frac{\varepsilon^2}{2}$. Así, cada iteración duplica el número de bits de cifras significativas. Por lo tanto, tenemos que $y_2 \approx \sqrt{x}$ con una precisión cuádruple, es decir, 113 bits de precisión.

Si $x \in [LDBL_{Min}, LDBL_{Max}]$, calculamos una primera aproximación de $\sqrt{x}$ con la función $sqrtl(x)$. Esta aproximación inicial tiene una precisión de 65 bits significativos. Por lo tanto, a diferencia del caso anterior, basta con una iteración del método de *Newton-Raphson* ($y_1 \approx \sqrt{x}$) para conseguir la precisión deseada de 113 bits.

Por otra parte, si x está fuera del rango de valores representables en un *double* o un *Long-double* utilizamos el siguiente algoritmo para el cálculo de la aproximación inicial de $\sqrt{x}$:

1. Expresamos x de la forma
$$x = F \cdot 2^n$$
donde $n \in \mathbb{Z}$ y $F \in \left[\frac{1}{2}, 1\right]$. Entonces
$$\sqrt{x} = \sqrt{F} \cdot \sqrt{2^n}$$
Empleamos la función auxiliar $frexpq(x, \&exp)$. Esta función retorna un valor entre $\frac{1}{2}$ y 1 y en la variable exp guarda el valor del exponente de x, es decir, $exp = n$.

2. Si exp es impar, calculamos $F = 2 \cdot F$ y $exp = exp - 1$.

   De esta forma siempre tendremos que el exponente es par, por lo que no tendremos que calcular $\sqrt{2^{exp}}$. Puesto que si exp es impar tendríamos que calcular $\sqrt{2^{2 \cdot k+1}}$ y este cálculo no es trivial.

   Entonces
   $$\sqrt{2^{exp}} = \sqrt{2^{2 \cdot k}} = 2^k \qquad k \in \mathbb{Z}$$
   Solo tenemos que dividir el exponente entre 2, al ser un número entero, esta operación no produce ningún error de propagación en el cálculo.

3. Calculamos $y_0 = sqrt(F)$

   Entonces
   $$y_0 = y \cdot \sqrt{2^{2 \cdot k}} = y \cdot 2^k$$

4. Calculamos $y_0 = y \cdot 2^k$ con la función auxiliar $scalbnq(y, k)$. La función $scalbnq$ calcula el resultado de $y \cdot 2^k$ mediante la manipulación del exponente, en lugar de realizar una exponenciación y una multiplicación.

Ahora tenemos que y_0 es una aproximación inicial de $\sqrt{x}$, con una precisión de 53 bits. Entonces, procedemos a la realización de 2 iteraciones del método de *Newton-Raphson* para conseguir la precisión cuádruple deseada en el cálculo de $\sqrt{x}$.

5.2. Raíz cúbica

La función $cbrtq(x)$ recibe un número real de cuádruple precisión y retorna el valor de la raíz cúbica de dicho número, con una precisión de 113 bits ($cbrtq(x) = \sqrt[3]{x}$). La raíz cúbica real de x es la raíz de la ecuación algebraica
$$y^3 - x = 0$$
El cálculo de $\sqrt[3]{x}$ consiste básicamente en 3 pasos: Primero la reducción del argumento x, en la forma $x = F \cdot 2^n$. Segundo, el cálculo de $\sqrt[3]{F}$, y por último la recuperación del valor de $\sqrt[3]{x}$ a partir de estos resultados.

Por tanto, lo primero que hacemos es extraer las potencias de 2, dejando la mantisa entre $\frac{1}{2}$ y 1. $x = F \cdot 2^{exp}$, con $\frac{1}{2} \leq F \leq 1 \quad$ y $\quad exp \in \mathbb{Z}$. Análogo a la raíz cuadrada.

Una vez hecha la extracción, calculamos la raíz cúbica de la mantisa, $y = \sqrt[3]{F}$. Este cálculo lo hacemos mediante una aproximación por polinomios de Chebyshev de la función $f(z) = \sqrt[3]{z}$.

$$\sqrt[3]{z} = \frac{1}{2}c_0 + \sum_{i=1}^{\infty} c_n T_n$$

Donde

$$c_n = \frac{2}{\pi}\int_{-1}^{1} \frac{\sqrt[3]{z}T_n(z)}{\sqrt{1-z^2}}dz \qquad n = 0, 1, 2, \cdots$$

Truncamos la serie en $n = 6$ y obtenemos la siguiente aproximación:

$$\begin{aligned} P_6(z) =& 0{,}13584464340920900529734z^5 - 0{,}63986917220457538402318z^4 \\ &+1{,}28755516703187515380 6z^3 - 1{,}48970833913572849578 91z^2 \\ &+1{,}33049612360136470925 2z + 0{,}37568280825958912391243 \qquad z \in \left[\frac{1}{2}, 1\right] \end{aligned}$$

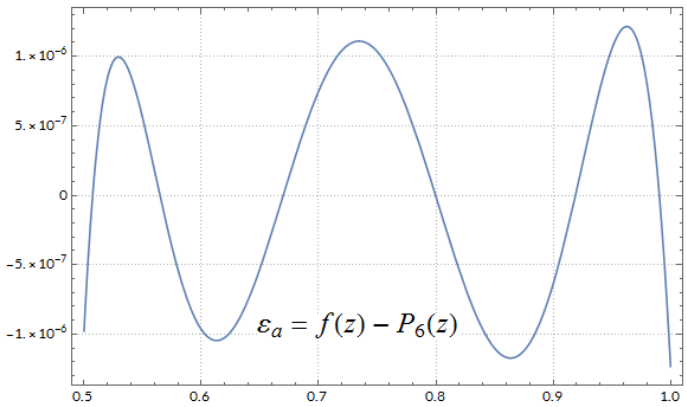

Figura 2: Error en la aproximación de $\sqrt[3]{z}$

Como podemos apreciar en la figura 2, la aproximación tiene un error relativo máximo de $-1{,}23 \cdot 10^{-6}$ para $z \in \left[\frac{1}{2}, 1\right]$.

$$\max_{z \in \left[\frac{1}{2}, 1\right]} |f(z) - P_6(z)| \approx |f(1) - P(1)| \approx 1{,}23 \cdot 10^{-6}$$

Por tanto, $y = P_6(F)$.

Por otra parte, podemos expresar el exponente de la forma $exp = 3k + r$, donde $k \in \mathbb{Z}$ y $r \in \{0, 1, 2\}$. Entonces

$$\sqrt[3]{x} = y \cdot \sqrt[3]{2^{(3k+r)}} = y \cdot \sqrt[3]{2^r} \cdot 2^k$$

El algoritmo para calcular una aproximación inicial de $\sqrt[3]{x}$ es el siguiente:

- Si $exp \geq 0$
  1. Calculamos $k = \left[\frac{exp}{3}\right]$
  2. Calculamos $r = exp - 3k$
     - 2.1. Si $r = 1$, calculamos $y = y \cdot \sqrt[3]{2}$
     - 2.2. Si $r = 2$, calculamos $y = y \cdot \sqrt[3]{4}$
- Si $exp < 0$
  1. $exp = -exp$
  2. Calculamos $k = \left[\frac{exp}{3}\right]$
  3. Calculamos $r = exp - 3k$
     - 3.1. Si $r = 1$, calculamos $y = y \cdot \sqrt[3]{\frac{1}{2}}$
     - 3.2. Si $r = 2$, calculamos $y = y \cdot \sqrt[3]{\frac{1}{4}}$
  4. $exp = -exp$

Una vez finalizado el algoritmo obtenemos $y = \sqrt[3]{F} \cdot \sqrt[3]{2^r}$. Por lo tanto, para tener una aproximación inicial de $\sqrt[3]{x}$ solo falta calcular $y \cdot 2^k$, este cálculo lo hacemos con la función *scalbnq*. $y_0 = scalbnq(y, k)$.

La aproximación inicial y_0, a diferencia del caso de la raíz cuadrada, no tiene una precisión de 53 bits debido a que $\sqrt[3]{F}$ está calculada con error relativo de $1{,}23 \cdot 10^{-6}$. Por tanto, hacen falta 3 iteraciones del método de *Newton-Raphson*, de la función $f(y) = y^3 - x$ con punto inicial y_0, para lograr tener una aproximación de $\sqrt[3]{x}$ con una precisión cuádruple.

$$y_1 = y_0 - \frac{y_0^3 - x}{3y_0^2}$$

En esta expresión tenemos: seis productos, dos sumas y una división. Por tanto, análogo a la raíz cuadrada, simplificamos la expresión para reducir el error de propagación en cada iteració.

$$y_1 = y_0 - \left(y_0 - \frac{x}{y_0^2}\right) \cdot \frac{1}{3} \tag{5.4}$$

Así, cada iteración requiere: dos productos, dos sumas y una división, ya que $\frac{1}{3}$ lo tenemos almacenado en la memoria.

$$y_2 = y_1 - \left(y_1 - \frac{x}{y_1^2}\right) \cdot \frac{1}{3}$$
$$y_3 = y_2 - \left(y_2 - \frac{x}{y_2^2}\right) \cdot \frac{1}{3}$$

Tomando $y_n = (1 + \varepsilon_n)\sqrt[3]{x}$ y sustituyendo en (5.4) encontramos que el error relativo de la (n+1)-ésima iteración es

$$\varepsilon_{n+1} = \varepsilon_n^2 \frac{1 + \frac{2}{3}\varepsilon_n}{(1 + \varepsilon_n)^2}$$

Así que, tenemos $y_3 \approx \sqrt[3]{x}$ con una precisión de 113 bits.

6. Funciones trigonométricas

6.1. Coseno

Esta función recibe un argumento x y retorna el valor de la función $\cos(x)$. A diferencia de la raíz cuadrada y la raíz cúbica, en esta función no podemos utilizar el método de *Newton-Raphson* para obtener una precisión de 113 bits, puesto que para aplicar dicho método necesitaríamos: una aproximación inicial $y_0 = \cos(x)$, la función $\arccos(y)$ y $\frac{d}{dy}(\arccos(y)) = -\frac{1}{\sqrt{1-y^2}}$.

$$y_1 = y_0 - \frac{\arccos(y_0) - x}{-\frac{1}{\sqrt{1-y^2}}}$$

No tenemos forma de obtener estos cálculos de manera elemental y por lo tanto, tenemos que aplicar un método alternativo.

La función $\cos(x)$ tiene una periodicidad de 2π y está definida $\forall x \in \mathbb{R}$. Se cumple la desigualdad $-1 \leq \cos(x) \leq 1$, es decir, no tenemos que preocuparnos de que se produzca desbordamiento en el cálculo.

Para implementar adecuadamente la función $\cos(x)$, necesitamos tomar el argumento de entrada y reducirlo a un intervalo simétrico cercano a 0 donde tengamos mejor controlada la precisión del cálculo.

La precisión de los valores de la función depende mucho de la precisión de la reducción del argumento. Por ello, expresamos los valores como suma de dos números. Donde el primer número corresponde a la parte más significativa y el segundo corresponde a la cola del valor.

Una vez tengamos la reducción del argumento, evaluamos una función trigonométrica (sin o cos) de este nuevo argumento y a partir de estos resultado,

recuperamos el valor de $\cos(x)$ del argumento inicial.

El método aplicado es el siguiente:

1. Si $|x| \geq \frac{\pi}{4}$, entonces hacemos una reducción del argumento x de la forma $y = x - k \cdot \frac{\pi}{2}$, con $y \in \left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$, $k \in \mathbb{Z}$. Expresamos y de la forma: $y = y_1 + y_2$, donde y_1 es la parte alta de y, e y_2 es la cola de y. Calculamos $n = k(mod4)$. Esta reducción del argumento la hacemos con la función rem_pio2q, dicha función hace la reducción del argumento y retorna el valor de n.

   Aplicando la propiedad

   $$\cos(A + B) = \cos(A)\cos(B) - \sin(A)\sin(B)$$

   Tenemos que

   - Si $n = 0$, entonces $\cos(x) = \cos\left(y_1 + y_2 + 0 \cdot \frac{\pi}{2}\right) = \quad \cos(y_1 + y_2)$.
   - Si $n = 1$, entonces $\cos(x) = \cos\left(y_1 + y_2 + 1 \cdot \frac{\pi}{2}\right) = -\sin(y_1 + y_2)$.
   - Si $n = 2$, entonces $\cos(x) = \cos\left(y_1 + y_2 + 2 \cdot \frac{\pi}{2}\right) = -\cos(y_1 + y_2)$.
   - Si $n = 3$, entonces $\cos(x) = \cos\left(y_1 + y_2 + 3 \cdot \frac{\pi}{2}\right) = \quad \sin(y_1 + y_2)$.

   La función *cosq_kernel* (*sinq_kernel*) calcula el valor de $\cos(\alpha)$ ($\sin(\alpha)$) para $|\alpha| \leq \frac{\pi}{4}$ con una precisión cuádruple.

   Por lo tanto

   1.1. Si $n = 0$, calculamos $cos(x) \approx cosq\_kernel(y_1, y_2)$.

   1.2. Si $n = 1$, calculamos $cos(x) \approx -sinq\_kernel(y_1, y_2, 1)$.

   1.3. Si $n = 2$, calculamos $cos(x) \approx -cosq\_kernel(y_1, y_2)$.

   1.4. Si $n = 3$, calculamos $cos(x) \approx sinq\_kernel(y_1, y_2, 1)$.

2. Si $|x| < \frac{\pi}{4}$ no necesitamos reducir el argumento, entonces $y_1 = x$ e $y_2 = 0$ ya que la cola de x es cero . Calculamos $cos(x) \approx cosq\_kernel(y_1, y_2)$.

6.2. Seno

Dado un argumento x esta función calcula el valor de $\sin(x)$ con una precisión cuádruple. Igual que en la función $\cos(x)$, en esta función no podemos aplicar el método de *Newton-Raphson*.

$\sin(x)$ es una función periódica de período 2π, está definida $\forall x \in \mathbb{R}$ y se cumple que

$$-1 \leq \sin(x) \leq 1$$

Por lo tanto, aplicando estas propiedades, dado un argumento x el cálculo de $\sin(x)$ consiste básicamente en 3 pasos numéricamente distinto: en primer lugar, necesitamos reducir el argumento al intervalo $\left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$ (en caso de ser necesario). El segundo paso es la evaluación de una función trigonométrica sobre el intervalo $\left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$ y por último, la reconstrucción de la función $\sin(x)$ a partir de estos resultados.

el método aplicado es el siguiente:

1. Si $|x| \geq \frac{\pi}{4}$, necesitamos una reducción del argumento x de la forma

$$y_1 + y_2 = x - k \cdot \frac{\pi}{2}$$

Calculamos $n = k(mod4)$ $(n = rem\_pio2q(x, y_1, y_2))$. Aplicando la propiedad

$$\sin(A + B) = \sin(A)\cos(B) + \cos(A)\sin(B)$$

Tenemos que

- Si $n = 0$, $\sin(x) = \sin\left(y_1 + y_2 + 0 \cdot \frac{\pi}{2}\right) = \sin(y_1 + y_2)$.
- Si $n = 1$, $\sin(x) = \sin\left(y_1 + y_2 + 1 \cdot \frac{\pi}{2}\right) = \cos(y_1 + y_2)$.
- Si $n = 2$, $\sin(x) = \sin\left(y_1 + y_2 + 2 \cdot \frac{\pi}{2}\right) = -\sin(y_1 + y_2)$.
- Si $n = 3$, $\sin(x) = \sin\left(y_1 + y_2 + 3 \cdot \frac{\pi}{2}\right) = -\cos(y_1 + y_2)$.

Por lo tanto,

1.1. Si $n = 0$, calculamos $\sin(x) \approx sinq\_kernel(y_1, y_2, 1)$.

1.2. Si $n = 1$, calculamos $\sin(x) \approx cosq\_kernel(y_1, y_2)$.

1.3. Si $n = 2$, calculamos $\sin(x) \approx -sinq\_kernel(y_1, y_2, 1)$.

1.4. Si $n = 3$, calculamos $\sin(x) \approx -cosq\_kernel(y_1, y_2)$.

2. Si $|x| < \frac{\pi}{4}$, entonces $y_1 = x$, $y_2 = 0$. Calculamos $sinq\_kernel(y_1, y_2, 0)$.

6.3. Funcion complementaria *cosq_kernel*

La función *cosq_kernel* es la función complementaria de $\cos(x)$. Una vez hecha la reducción del argumento esta es la función encargada de calcular la aproximación $\cos(y)$, en el intervalo $\left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$, con una precisión cuádruple. Esta función recibe dos argumentos: y_1 e y_2, donde $|y_1| \leq \frac{\pi}{4}$ es el argumento al cual queremos calcular el *coseno* e y_2 es la cola de y_1.

La serie de Taylor de $\cos(y)$ es:

$$\cos(y) = \sum_{n=0}^{\infty} \frac{(-1)^n y^{2n}}{(2n)!}$$

La serie de Taylor es una muy buena aproximación de $\cos(y)$ para ángulos muy pequeños, alrededor del cero, pero su error aumenta casi de forma exponencial a medida que nos alejamos del cero.

La serie de Chebyshev, viene dada por:

$$\cos(y) = \frac{c_0}{2} + \sum_{k=0}^{\infty} c_k T_k(y)$$

Donde

$$c_k = \frac{2}{\pi} \int_{-1}^{1} \frac{\cos(y) T_k(y)}{\sqrt{1-y^2}} dy$$

Haciendo el cambio de variable $y = \cos(\theta)$, tenemos que

$$c_k = \frac{2}{\pi} \int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \cos(\cos(\theta)) \cos(ky) dy = (-1)^k 2 \cdot J_{2k}(1)$$

$J_{2k}(1)$ es la función de Bessel de primera especie. $T_k(y)$ es el polinomio de Chebyshev de primera especie de grado n[9]. Entonces

$$\cos(y) = J_0(1) + 2\sum_{k=1}^{\infty} (-1)^k J_{2k}(1) T_{2k}(y)$$

Podemos expresar esta serie como una serie de potencias.

$$\cos(y) = \sum_{i=0}^{\infty} a_i x^{2i}$$

La precisión de la aproximación de la serie de Chebyshev depende de la velocidad con la que $J_{2k}(1)$ tiende a cero

$$\left| J_{2k}(1) \approx \frac{1}{4^k (2k)!} \right|$$

La serie de Chebyshev tiene una mejor precisión en todo el intervalo y su error es uniforme. Por lo tanto, utilizaremos la serie de Chebyshev para aproximar $\cos(y)$.

Como en anteriormente, separamos diversas situaciones:

1. Si $|y_1 + y_2| < 2^{-57}$. El desarrollo de Taylor de $\cos(y) = 1 - \frac{y^2}{2} + \frac{y^4}{24} \cdots$. tenemos $\varepsilon_a(y) = |cos(y) - 1| \leq \frac{(2^{-57})^2}{2} = 2^{-113} \quad \forall |y| \leq 2^{-57}$. Entonces calculamos $\cos(y_1) \approx= 1$. Este calculo tiene una precisión de 113 bits.

2. Si $2^{-57} \leq |y_1 + y_2| < 0{,}1484375$, el argumento es lo suficientemente pequeño como para aproximar $\cos(y)$ mediante una aproximación polinomial de Chebyshev de grado 16 y obtener gran precisión.

   Truncamos la serie de Chebyshev en $n = 8$:

$$\cos(y) = a_0 + a_1 y^2 + a_2 y^4 + a_3 y^6 + a_4 y^8 + a_5 y^{10} + a_6 y^{12} + a_7 y^{14} + a_8 y^{16}$$

$$y \in \left[2^{-57}, 0{,}1484375\right)$$

   Donde

   - $a_0 = \quad 1{,}00000000000000000000000000000000$
   - $a_1 = -4{,}99999999999999999999999999999759 \cdot 10^{-1}$
   - $a_2 = \quad 4{,}16666666666666666666666666651287795 \cdot 10^{-2}$
   - $a_3 = -1{,}38888888888888888888888742314300284 \cdot 10^{-3}$
   - $a_4 = \quad 2{,}48015873015873015867694002851118210 \cdot 10^{-5}$
   - $a_5 = -2{,}75573192239858811636614709689300351 \cdot 10^{-7}$
   - $a_6 = \quad 2{,}08767569877762248667431926878073669 \cdot 10^{-9}$
   - $a_7 = -1{,}14707451049343817400420280514614892 \cdot 10^{-11}$
   - $a_8 = \quad 4{,}77810092804389587579843296923533297 \cdot 10^{-14}$

   Aplicando la fórmula de la aproximación de Chebyshev, vemos que esta aproximación tiene un error absoluto aproximado de $1{,}46 \cdot 10^{-36} (< 2^{-113})$.

$$\varepsilon_a(y) \leq \frac{(0{,}1484375)^{18}}{2^{17} \cdot 18!} \approx 1{,}46 \cdot 10^{-36} \qquad y \in \left[2^{-57}, 0{,}1484375\right)$$

   Si evaluamos este polinomio de forma lineal, tenemos que realizar 80 productos y 8 sumas. Esto supondría un coste computacional muy alto y además, tendremos mucho error de propagación. Si $y = \bar{y} + \varepsilon$, entonces el error en la función es

$$\cos(y + \varepsilon) = \cos(\bar{y})\cos(\varepsilon) + \sin(\bar{y})\sin(\varepsilon) \overset{\varepsilon \ll 1}{\approx} \cos(\bar{y}) - \varepsilon \sin(\bar{y})$$

   Y el error de propagación de los productos es $\dfrac{80 \cdot \varepsilon}{|y|}$

   Dado que el polinomio de grado 16 que aproxima a $\cos(y)$ solo tiene potencias de y pares, definimos la variable

$$z = y_1 \cdot y_1$$

   Así, tenemos que

$$\cos(y_1) = a_0 + a_1 z + a_2 z^2 + a_3 z^3 + a_4 z^4 + a_5 z^5 + a_6 z^6 + a_7 z^7 + a_8 z^8$$

Lo que supone un ahorro computacional. Además, utilizamos la evaluación del polinomio por el método de Horner. Calculamos $\cos(y_1)$

$$\cos(y_1) \approx a_0+(z\cdot(a_1+z\cdot(a_2+z\cdot(a_3+z\cdot(a_4+z\cdot(a_5+z\cdot(a_6+z\cdot(a_7+a_8\cdot z)))))))$$

De esta forma, solo tenemos que hacer: 9 multiplicaciones y 8 sumas, para evaluar el polinomio de grado 16, es decir, esta forma de evaluar el polinomio es muy eficiente. $\cos(y_1)$ tiene una precisión de 113 bits.

3. Si $0{,}1484375 \leq |y_1 + y_2| \leq \frac{\pi}{4}$, entonces, para no usar un polinomio de grado muy elevado con el fin de obtener una precisión cuádruple, buscamos l y h tal que

$$y_1 = l + h_j$$

Donde h_j son puntos equidistantes del intervalo $\left[0{,}1484375, \frac{\pi}{4}\right]$

$$h_j = 0{,}1484375 + \frac{j}{128} \qquad j = 0, 1, \cdots, 82$$

Por lo tanto h_j tiene 83 posibles resultados. Ya que $h_{83} > \dfrac{\pi}{4}$. Se cumple que $h_j \leq y_1 \leq h_{j+1}$, entonces $|l| < \frac{1}{256}$ [10].

Entonces, se tiene

$$\cos(y_1) = \cos(l + h_j)$$
$$\cos(l + h_j) = \cos(h_j) \cdot \cos(l) - \sin(h_j) \cdot \sin(l)$$

3.1. Usamos la función $signbitq(y_1)$. Esta función retorna el bit del signo de y_1

3.1.1. Si $signbitq(y_1) = 1$, definimos $y_1 = -y_1$ e $y_2 = y_2$, ya que $\cos(-\theta) = \cos(\theta)$. De esta forma solo trabajamos con valores positivos.

3.2. Calculamos h_j y buscamos $\cos(h_j)$ y $\sin(h_j)$ en las tablas precalculadas. Dichas tablas están estructuradas de la siguiente forma: Tenemos 83 bloques de cuatro filas, un bloque para cada posible valor de h_j. En la primera fila, de cada bloque, tenemos los primeros 113 bits de $\cos(h_j)$ ($\cos(h_{j\_hi})$), en la segunda fila están los últimos 113 bits de $\cos(h_j)$ ($\cos(h_{j\_lo})$), en la tercera tenemos los primeros 113 bits de $\sin(h_j)$ ($\sin(h_{j\_hi})$) y en la cuarta fila los últimos 113 bits de $\sin(h_j)$ ($\sin(h_{j\_lo})$).

De esta forma, obtenemos mayor precisión en el cálculo de $\cos(h_j)$ y $\sin(h_j)$.

$$\cos(h_j) = \cos(h_{j\_hi}) + \cos(h_{j\_lo})$$
$$\sin(h_j) = \sin(h_{j\_hi}) + \sin(h_{j\_lo})$$

3.3. Calculamos $l = y_2 - (y_1 - h_j)$, $z = y_1 \cdot y_1$

3.4. Calculamos $\sin(l)$. Utilizamos la serie de Chebyshev de $\sin(l)$

$$\sin(l) = \frac{c_0}{2} + \sum_{k=1}^{\infty} c_k T_k(l)$$

Truncamos la serie en $k = 11$

$$\sin(l) = s_0 l + s_1 l^3 + s_2 l^5 + s_3 l^7 + s_4 l^9 + s_5 l^{11} \qquad l \in \left[0, \frac{1}{256}\right] \tag{6.1}$$

Donde

- $s_1 = -1{,}66666666666666666666666666666659 \cdot 10^{-1}$
- $s_2 = \ \ 8{,}33333333333333333333333333146298442 \cdot 10^{-3}$
- $s_3 = -1{,}98412698412698412697726277416810661 \cdot 10^{-4}$
- $s_4 = \ \ 2{,}75573192239848624174178393552189149 \cdot 10^{-6}$
- $s_5 = -2{,}50521016467996193495359189395805639 \cdot 10^{-8}$

La estimación del error de la aproximación es

$$\varepsilon_a(l) \leq \frac{(1/256)^{13}}{2^{12} \cdot 13!} \approx 1{,}93 \cdot 10^{-45} \qquad |l| < \frac{1}{256}$$

Calculamos $\sin(l) \approx l \cdot (s_0 + z \cdot (s_1 + z \cdot (s_2 + z \cdot (s_3 + z \cdot (s_4 + s_5 \cdot z)))))$

3.5. Truncamos la serie de Chebyshev de $\cos(l)$ en $k = 10$. Aplicando la fórmula de la estimación del error tenemos que $\varepsilon_a(l) \leq 1{,}28 \cdot 10^{-41}$

$$\cos(l) = c_0 + c_1 l^2 + c_2 l^4 + c_3 l^6 + c_4 l^8 + c_5 l^{10} \qquad |l| < \frac{1}{256} \tag{6.2}$$

Donde

- $c_0 = \ \ 1{,}00000000000000000000000000000000$
- $c_1 = -5{,}00000000000000000000000000000000 \cdot 10^{-1}$
- $c_2 = \ \ 4{,}16666666666666666666666666556146073 \cdot 10^{-2}$
- $c_3 = -1{,}38888888888888888888309442601939728 \cdot 10^{-3}$
- $c_4 = \ \ 2{,}48015873015862382987049502531095061 \cdot 10^{-5}$
- $c_5 = -2{,}75573112601362126593516899592158083 \cdot 10^{-7}$

Para obtener una mayor precisión calculamos $\cos(l) - 1$, en lugar de $\cos(l)$. Ya que para $|l| < \frac{1}{256}$ $\cos(l)$ es muy cercano a 1 y esto provoca inestabilidad númerica en el cálculo.

$$\cos_1 = \cos(l) = z \cdot (c_1 + z \cdot (c_2 + z \cdot (c_3 + z \cdot (c_4 + c_5 \cdot z))))$$

3.6. Recuperamos la función, es decir, calculamos el valor de $\cos(y_1)$ a partir de los resultados anteriores.

$$\begin{aligned}\cos(y_1) &= \cos(h_j)\cdot\cos(l) - \sin(h_j)\cdot\sin(l)\\ &= (\cos(h_{j\_hi}) + \cos(h_{j\_lo}))\cdot(\cos_1 + 1) - (\sin(h_{j\_hi}) + \sin(h_{j\_lo}))\cdot\sin(l)\end{aligned}$$

Como $\cos(h_{j\_lo}) \ll 1$ y $\sin(h_{j\_lo}) \ll 1$, entonces $cosq\_kernel(y_{1,y}\, 2)$ calcula

$$\cos(y_1) \approx \cos(h_{j\_hi})\cdot\cos_1 + \cos(h_{j\_hi}) + \cos(h_{j\_lo}) - \sin(h_{j\_hi})\cdot\sin(l)$$

Es decir, $cosq\_kernel(y_{1,y}\, 2)$ calcula el valor $\cos(y_1)$ con una precisión cuádruple.

6.4. Función complementaria *sinq_kernel*

La función $sinq\_kernel(y_1, y_2, c)$ recibe 3 argumentos: y_1, y_2 y c. Donde $y_1 \in \left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$ es una variable, y_2 es la cola de y_1 y c es una constante que tiene valor 0 o 1. *sinq_kernel* calcula el valor de $\sin(y_1)$ con una precisión de 113 bits.

El método aplicado es el siguiente:

1. Si $|y_1 + y_2| < 2^{-57}$, análogo al cálculo de $\cos(y_1)$, calculamos $\sin(y_1) \approx y_1$.
2. Si $2^{-57} \leq |y_1 + y_2| < 0{,}14843375$, aproximando $\sin(y)$ por la serie de Chebyshev para $k = 17$, obtenemos la precisión deseada.

   $$\sin(y) = y + m_1 y^3 + m_2 y^5 + \cdots + m_8 y^{17} \qquad y \in \left[2^{-57}, 0{,}1484375\right]$$

   Donde

   - $m_1 = -1{,}66666666666666666666666666666538 \cdot 10^{-1}$
   - $m_2 = 8{,}33333333333333333333333333307532934 \cdot 10^{-3}$
   - $m_3 = -1{,}98412698412698412698412534478712057 \cdot 10^{-4}$
   - $m_4 = 2{,}75573192239858906520896496653095890 \cdot 10^{-6}$
   - $m_5 = -2{,}50521083854417116999224301266655662 \cdot 10^{-8}$
   - $m_6 = 1{,}60590438367608957516841576404938118 \cdot 10^{-10}$
   - $m_7 = -7{,}64716343504264506714019494041582610 \cdot 10^{-13}$
   - $m_8 = 2{,}81068754939739570236322404393398135 \cdot 10^{-15}$

   Definimos $z = y_1 \cdot y_1$, entonces calculamos

   $$\sin(y_1) \approx y_1 + (y_1 \cdot (z \cdot (m_1 + z \cdot (m_2 + z(\cdots(m_7 + m_8 \cdot z))))))$$

3. Si $|y_1 + y_2| \geq 0{,}1484375$, igual que en la función *cosq_kernel*, buscamos l y h tal que

   $$y_1 = l + h_j$$

3.1. Calculamos h_j y buscamos $\cos(h_j)$ y $sin(h_j)$ en las tablas precalculadas.

3.2. Calculamos $sign = sign(x)$, $y_1 = |y_1|$

3.3. Si $c = 1$

3.3.1. Si $sign = 1$, calculamos $l = -y_2 - (h_j - y_1)$

3.3.2. Si $sign = 0$, calculamos $l = y_2 - (h_j - y_1)$

3.4. Si $c = 0$, calculamos $l = y_1 - h_j$

3.5. Calculamos $\sin(l)$ utilizando la aproximación de la serie de Chebyshev de grado 11 (6.1), calculada anteriormente

3.5.1. Si $sign = 1$, entonces $\sin(l) = -\sin(l)$

3.6. Calculamos $\cos_1 = \cos(l) - 1$ mediante la aproximación de la serie de Chebyshev de grado 10 (6.2)

3.7. Recuperamos el valor de $sin(y_1)$

$$\begin{aligned}\sin(y_1) &= \sin(l + h_j) = \sin(l) \cdot \cos(h_j) + \cos(l) \cdot \sin(h_j) \\ &= \sin(l)\,(\cos(h_{j\_hi}) + \cos(h_{j\_lo})) + (\cos_1 + 1)\,(\sin(h_{j\_hi}) + \sin(h_{j\_lo}))\end{aligned}$$

$$\cos(h_{j\_lo}) \ll 1, \quad \sin(h_{j\_lo}) \ll 1 \Longrightarrow$$

$$\Longrightarrow \sin(y_1) \approx \sin(l) \cdot \cos(h_{j\_hi}) + \cos_1 \cdot \sin(h_{j\_hi}) + \sin(h_{j\_hi})$$

6.5. Tangente

La tangente de un ángulo está definida matemáticamente por el cociente entre $\cos(x)$ y $\sin(x)$

$$\tan(x) = \frac{\sin(x)}{\cos(x)}$$

Sin embargo, no podemos utilizar una aproximación de $\sin(x)$ y otra de $\cos(x)$ con una precisión de 113 bits y hacer la división para calcular $\tan(x)$. Ya que esto sería muy costoso a nivel operacional. Además, no tendríamos la precisión cuádruple deseada en el cálculo de $\tan(x)$. Por lo tanto, utilizamos otro método para realizar este cálculo.

En la aproximación de $\tan(x)$ con una precisión de 113 bits, igual que en la aproximación de $\sin(x)$ y $\cos(x)$, están involucrados 3 pasos numéricamente distintos:

- Reducción del argumento al intervalo $\left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$.
- Evaluación de $\tan(x)$ sobre el intervalo $\left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$.

- Recuperación de $\tan(x)$ a partir de los resultados de los pasos anteriores.

El algoritmo para calcular la $\tan(x)$ con una precisión cuádruple es el siguiente:

1. Si $x = NaN$ o INF Calculamos $x - x$, esto nos genera una advertencia.
2. Si $|x| > \dfrac{\pi}{4}$, entonces necesitamos la reducción del argumento. Calculamos

$$n = rem\_pio2q(x, y_1, y_2)$$

   Entonces

$$\tan(x) = \tan\left(y_1 + y_2 + n \cdot \frac{\pi}{2}\right) \qquad n = 0, 1, 2, 3$$

   Utilizando las propiedades de vistas anteriormente de $\sin(x)$ y $\cos(x)$., tenemos:

   - Si $n = 0$, $\tan\left(y_1 + y_2 + 0 \cdot \dfrac{\pi}{2}\right) = \dfrac{\sin(y_1 + y_2)}{\cos(y_1 + y_2)} = \tan(y_1 + y_2)$
   - Si $n = 1$, $\tan\left(y_1 + y_2 + 1 \cdot \dfrac{\pi}{2}\right) = \dfrac{\sin(y_1 + y_2 + \frac{\pi}{2})}{\cos(y_1 + y_2 + \frac{\pi}{2})} = -\dfrac{1}{\tan(y_1 + y_2)}$
   - Si $n = 2$, $\tan\left(y_1 + y_2 + 2 \cdot \dfrac{\pi}{2}\right) = \dfrac{\sin(y_1 + y_2 + \pi)}{\cos(y_1 + y_2 + \pi)} = \tan(y_1 + y_2)$
   - Si $n = 3$, $\tan\left(y_1 + y_2 + 3 \cdot \dfrac{\pi}{2}\right) = \dfrac{\sin(y_1 + y_2 + \frac{3\pi}{2})}{\cos(y_1 + y_2 + \frac{3\pi}{2})} = -\dfrac{1}{\tan(y_1 + y_2)}$

   Ahora tenemos que el argumento es menor que $\dfrac{\pi}{4}$, por lo cual podemos utilizar la función *tanq_kernel*.

   2.1. Si n es par, $\tan(x) = \tan(y_1 + y_2)$, entonces definimos $s = 1$.

   2.2. Si n es impar, $\tan(x) = -\frac{1}{\tan(y_1+y_2)}$, entonces definimos $s = -1$.

   2.3. Calculamos $tanq\_kernel(y_1, y_2, s)$.

3. Si $|x| \leq \dfrac{\pi}{4}$, entonces tenemos: $y_1 = x$, $y_2 = 0$ y $s = 1$. Calculamos $tanq\_kernel(y_1, y_2, s)$.

6.6. Función complementari *tanq_kernel*

La función $tanq\_kernel(y_1, y_2, s)$ calcula el valor de la tangente en el intervalo $\left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$ con una precisión de 113 bits. Esta función recibe 3 argumentos y_1, y_2 y s. Donde $|y_1| \leq \frac{\pi}{4}$, y_2 es la cola de y_1 y s es un entero que tiene valor 1 o -1 e indica si se tiene que calcular $\tan(y_1 + y_2)$ o $-\frac{1}{\tan(y_1+y_2)}$.

Dado que la tangente tiene la propiedad que $\tan(-\theta) = -\tan(\theta)$, solo tenemos que considerar valores positivos para los cálculos.

1. Si $|y_1 + y_2| < 2^{-57}$. Por lo visto anteriormente $\cos(y) \approx 1$ y $\sin(y) \approx y$, calculamos $\tan(y_1) \approx y_1$.

2. Si $y_1 < 0$, definimos $sign = -1$, $y_1 = -y_1$ e $y_2 = -y_2$

3. Si $2^{-57} \leq y_1 + y_2 < \dfrac{\pi^2}{16}$, para el cálculo de $\tan(y)$ utilizamos una aproximación *minimax* racional de la forma

$$\tan(y) = y + \frac{y^3}{3} + y^5 \cdot R(y^2) \qquad [11]$$

Esto nos garantiza mejor precisión para ángulos cercanos a cero, ya que los primeros términos tienen el mayor efecto. En este caso, utilizamos una aproximación racional $R(y^2)$. Ya que de esta forma podemos obtener una mayor precisión, respecto a una aproximación polinomial, con menos operaciones. Esto supone un ahorro computacional.

$$R(y^2) = \frac{P(y^2)}{Q(y^2)} = \frac{\sum_{i=0}^{4} p_i y^{2i}}{\sum_{i=0}^{5} q_i y^{2i}} \qquad y \in \left[2^{-57}, \frac{\pi^2}{16}\right]$$

Donde

- $p_0 = -1{,}81301471174358343774236328433685588939 3 \cdot 10^7$
- $p_1 = \ \ 1{,}32076796000897222431274007508325924761 8 \cdot 10^6$
- $p_2 = -2{,}62677547825583818246865182186329902395 6 \cdot 10^4$
- $p_3 = \ \ 1{,}76457335648850493541541138368715019931 5 \cdot 10^2$
- $p_4 = -3{,}33326776382217869079467897897980352609 2 \cdot 10^{-1}$
- $q_0 = \ \ 1{,}00000000000000000000000000000000000000$
- $q_1 = -1{,}35976103380768757830677246325371004201 0 \cdot 10^8$
- $q_2 = \ \ 6{,}49437063065689317566672931306511319478 4 \cdot 10^7$
- $q_3 = -4{,}18078767223792747550553684916872938678 2 \cdot 10^6$
- $q_4 = \ \ 8{,}03164376510617004013996662298091462152 1 \cdot 10^4$
- $q_5 = -5{,}32313127191247569515712787556066737859 7 \cdot 10^2$

Queremos calcular $\tan(y_1 + y_2)$. Tenemos la siguiente propiedad

$$\tan(y_1 + y_2) \overset{y_2 \ll 1}{\approx} \tan(y_1) + \sec^2(y_1) \cdot y_2 \approx \tan(y_1) + (1 + y_1^2) \cdot y_2$$

Para tener una mayor precisión en este cálculo, definimos: $r = y_1^3 \cdot R(y_1^2)$. Entonces

$$\tan(y_1 + y_2) = y_1 + \frac{y_1^3}{3} + y_1^2(r + y_2) + y_2$$

4. Si $y_1 + y_2 \geq \dfrac{\pi^2}{16}$.

   Definimos $z = \dfrac{\pi_{hi}}{4} - y_1$, $w = \dfrac{\pi_{lo}}{4} - y_2$. Entonces $y_1 = z + w$ e $y_2 = 0$. Es decir, hacemos hecho $y_1 + y_2 = \frac{\pi}{4} - (y_1 + y_2)$.

   Por lo tanto, tendremos que calcular

$$\begin{gathered} \tan(y_1 + y_2) = \tan\left(\frac{\pi}{4} - y_1\right) = \frac{1 - \tan(y_1)}{1 + \tan(y_1)} \\ \Longrightarrow \tan(y_1 + y_2) = 1 - 2 \cdot \left(tan(y_1) - \frac{\tan^2(y_1)}{1 + \tan(y_1)}\right) \end{gathered} \tag{6.3}$$

   Con este cambio de variable tenemos que $y_1 + y_2 \leq \frac{\pi^2}{16}$, de esta forma podemos utilizar la aproximación *minimax* racional para calcular $\tan(y_1 + y_2)$.

5. Calculamos

$$\begin{aligned} z &= y_1 \cdot y_1 \\ p &= p_0 + z \cdot (p_1 + z \cdot (p_2 + z \cdot (p_3 + p_4 \cdot z))) \\ q &= q_0 + z \cdot (q_1 + z \cdot (q_2 + z \cdot (q_3 + z \cdot (q_4 + z)))) \\ R &= \frac{p}{q} \\ r &= z \cdot y_1 \cdot R \end{aligned}$$

6. Calculamos $\tan = y_1 + \dfrac{1}{3} z \cdot y_1 + z \cdot (r + y_2) + y_2$. Así obtenemos

$$\tan = \begin{cases} \tan(y_1 + y_2) & \text{si } y_1 + y_2 < \dfrac{\pi^2}{16} \\ \\ \tan(y_1) & \text{si } y_1 + y_2 \geq \dfrac{\pi^2}{16} \end{cases}$$

7. Si inicialmente teníamos que $y_1 + y_2 \geq \dfrac{\pi}{4}$, entonces tenemos que calcular (6.3)

$$\tan = s - 2 \cdot \left(\tan - \frac{\tan \cdot \tan}{s + \tan}\right)$$

   - Si $s = 1$:

$$\tan = 1 - 2 \cdot \left(\tan - \frac{\tan \cdot \tan}{1 + \tan}\right) = \tan(y_1 + y_2)$$

- Si $s = -1$:

$$\tan = -1 - 2 \cdot \left(\tan - \frac{\tan \cdot \tan}{-1 + \tan} \right) = \frac{-1 + \tan - 2\tan + 2\tan^2 - 2\tan^2}{1 - \tan}$$
$$= \frac{-1 - \tan}{1 - \tan}$$
$$\Longrightarrow \tan = -\frac{1}{\tan(y_1 + y_2)}$$

8. Si $sign = -1$, calculamos $\tan = -\tan$

Después de todo el proceso tenemos el valor de la tangente en el intervalo $\left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$, $\tan(y_1 + y_2) \approx \tan$ con una precisión de 113 bits.

7. Función exponencial

Queremos calcular $f(x) = e^x$ con una precisión cuádruple. $f(x)$ está definida $\forall x \in \mathbb{R}$, pero como trabajamos con números de punto flotante con una representación en 128 bits, tenemos que $f(x)$ está definida para x tal que

$$x_{min} = ln\left(\frac{2^{2-2^{14}}}{2^{p-1}}\right) \leq x \leq ln\left(\left(1 - \frac{1}{2^p}\right) \cdot 2^{2^{14}}\right) = x_{max}$$

$$x_{min} \approx -11432{,}7696 \quad \text{y} \quad x_{max} \approx 11356{,}5234$$

Ya que si $x \geq x_{max}$, entonces se tiene

$$e^x > e^{x_{max}} = e^{ln(FLT128\_MAX)} = FLT128\_MAX$$

Por lo tanto, se produce desbordamiento. Si $x < x_{min}$ se genera subdesbordamiento.

Trabajaremos con pares de números, donde el primer número es la parte alta y el segundo corresponde a la parte baja del número. Los primeros 93 bits corresponden a la parte alta del número, solo cogemos 93 bits para no añadir error de redondeo en las operaciones aritméticas.

Dado un argumento $x \in [x_{min}, x_{max}]$, lo reducimos a un intervalo mas pequeño cercano al origen donde podamos calcular con precisión e^x. Hacemos las siguientes reducciones:

$$1) \quad x^{(1)} = x - n \cdot ln\,(2) \tag{7.1}$$
$$2) \quad x^{(2)} = x^{(1)} - u_i^{(1)}$$
$$3) \quad x^{(3)} = x^{(2)} - u_j^{(2)}$$

Escogemos n de forma que se cumpla $-\frac{1}{2}ln(2) < x^{(1)} < \frac{1}{2}ln(2)$.

$u_i^{(1)}$ y $u_j^{(2)}$ son valores de unas tablas fijas almacenadas en la memoria. Estos valores se expresan de la forma

$$u_i^{(1)} = v_i^{(1)} + w_i^{(1)} \quad \text{y} \quad u_j^{(2)} = v_j^{(2)} + w_j^{(2)}$$

Donde $v_i^{(1)}$ es la parte alta de $u_i^{(1)}$, $w_i^{(1)}$ es la cola de $u_i^{(1)}$, $v_j^{(2)}$ es la parte alta de $u_j^{(2)}$ y $w_j^{(2)}$ es la cola de $u_j^{(2)}$.

La tabla $\left\{u_i^{(1)}\right\}$ cubre el intervalo $\left(-\frac{1}{2}ln(2), \frac{1}{2}ln(2)\right)$ con números con representación binaria exacta y que están más o menos equidistantes.

$$u_i^{(1)} \simeq i * \Delta^{(1)} \qquad \text{con} \qquad \Delta^{(1)} = \frac{1}{256}$$

Tenemos que $i \in \mathbb{Z}$ tal que $-\lceil ln(2) \cdot \Delta^{(1)} \rceil \leq i \leq \lceil ln(2) \cdot \Delta^{(1)} \rceil$, es decir, $i \in [-89, 89]$.
Donde $\lceil x \rceil$ denota la función techo. $\lceil x \rceil := \text{mín}\{k \in \mathbb{Z} | x \leq k\}$.

Dado $x^{(1)}$, i se elige de modo que $-\frac{1}{2}\overline{\Delta}^{(1)} < x^{(2)} < \frac{1}{2}\overline{\Delta}^{(1)}$. Donde $\overline{\Delta}^{(1)}$ es ligeramente mayor que $\Delta^{(1)}$ $\left(\overline{\Delta}^{(1)} = \Delta^{(1)} + \delta\right)$.

La tabla $\left\{u_j^{(2)}\right\}$ cubre el intervalo $\left(-\frac{1}{2}\overline{\Delta}^{(1)}, \frac{1}{2}\overline{\Delta}^{(1)}\right)$ con números más o menos equidistantes.

$$u_j^{(2)} \simeq j * \Delta^{(2)} \qquad \text{con} \qquad \Delta^{(2)} = \frac{1}{256} \cdot \frac{1}{128} = \frac{1}{2^{15}} \qquad j \in \mathbb{Z}$$

Tenemos que $-\lceil \frac{\overline{\Delta}^{(1)}}{2\Delta^{(2)}} \rceil \leq j \leq \lceil \frac{\overline{\Delta}^{(1)}}{2\Delta^{(2)}} \rceil \Rightarrow j \in [-65, 65]$.

Dado $x^{(2)}$, j se elige tal que $-\frac{1}{2}\overline{\Delta}^{(2)} < x^{(3)} < \frac{1}{2}\overline{\Delta}^{(2)}$. Donde $\overline{\Delta}^{(2)} = \Delta^{(2)} + \delta$.
Tenemos que

$$\begin{aligned} x^{(3)} &= x^{(2)} - u_j^{(2)} = x^{(1)} - u_i^{(1)} - u_j^{(2)} \\ &= x - n \cdot ln(2) - u_i^{(1)} - u_j^{(2)} \end{aligned}$$

Por lo tanto

$$\begin{aligned} x &= x^{(3)} + n \cdot ln(2) + u_i^{(1)} + u_j^{(2)} \\ &= x^{(3)} + n \cdot ln(2) + v_i^{(1)} + w_i^{(1)} + v_j^{(2)} + w_j^{(2)} \end{aligned}$$

Entonces aproximamos e^x como

$$\begin{aligned} e^x &= e^{(x^{(3)} + n \cdot ln(2) + v_i^{(1)} + w_i^{(1)} + v_j^{(2)} + w_j^{(2)})} \\ &= e^{x^{(3)}} \cdot e^{n \cdot ln(2)} \cdot e^{(v_i^{(1)} + w_i^{(1)})} \cdot e^{(v_j^{(2)} + w_j^{(2)})} \end{aligned} \tag{7.2}$$

Junto a las 4 tablas $\left\{v_i^{(1)}\right\}$, $\left\{w_i^{(1)}\right\}$, $\left\{v_j^{(2)}\right\}$ y $\left\{w_j^{(2)}\right\}$, también guardamos otras 2 tablas

$$\begin{aligned}\left\{y_i^{(1)}\right\} \quad & y_i^{(1)} = e^{u_i^{(1)}} = e^{(v_i^{(1)}+w_i^{(1)})} \\ \left\{y_j^{(2)}\right\} \quad & y_j^{(2)} = e^{u_j^{(2)}} = e^{(v_j^{(2)}+w_j^{(2)})}\end{aligned}$$

Estas tablas tienen la propiedad de que los puntos $u_i^{(1)}$ y $u_j^{(2)}$ son elegidos de tal manera que $y_i^{(1)}$ e $y_j^{(2)}$ tengan mantisas muy cortas, es decir, tengan muchos ceros en los últimos bits de la mantisa. Para preparar estas tablas, consideramos (v, w, y) tales que $y = e^{v+w}$. El valor de $v + w$ es conocido e y es expresable exactamente por un número de punto flotante cuya mantisa tiene muchos ceros en los últimos bits[12].

Esto se puede hacer a partir de una aproximación de $v + w$: Calculamos la aproximación de la exponencial de $v + w$, redondeamos el resultado a 113 bits y asignamos el resultado a y. Calculamos $\log(y)$ con una precisión lo suficientemente alta, dividimos el resultado de la mantisa en 2 partes y asignamos la parte más significativa a v y la menos significativa a w. Notemos que para la construcción de las tablas necesitamos utilizar un programa de cálculo externo.

Tenemos que

$$e^x = e^{x^{(3)}} \cdot 2^n \cdot y_i^{(1)} \cdot y_j^{(2)} = z + z\left(e^{x^{(3)}} - 1\right)$$

Donde $z = 2^n \cdot y_i^{(1)} \cdot y_j^{(2)}$. Dado que z es producto de números con muchos ceros en la mantisa, z es un número máquina, es decir, es expresable de forma exacta. Por lo tanto se puede calcular sin ningún error de redondeo.

La expresión $e^{x^{(3)}} - 1$ la calculamos mediante una aproximación *minimax* $P(x)$, $x \in \left(-\frac{1}{2}\overline{\Delta}^{(2)}, \frac{1}{2}\overline{\Delta}^{(2)}\right)$.Calculamos y guardamos en la memoria

$$\begin{aligned}P(x) = x + 0{,}5x^2 &+ 1{,}66666666666666666666666666666683 \cdot 10^{-1}x^3 \\ &+ 4{,}16666666666666666666654902320001674 \cdot 10^{-2}x^4 \\ &+ 8{,}33333333333333333333314659767198461 \cdot 10^{-3}x^5 \\ &+ 1{,}38888888889899438565058018857254025 \cdot 10^{-3}x^6 \\ &+ 1{,}98412698413981650382436541785404286 \cdot 10^{-4}x^7\end{aligned}$$

Esta aproximación *minimax* tiene un error relativo $\varepsilon_a < 5{,}69 \cdot 10^{-46}$

$$\varepsilon_a < \frac{\left(\frac{1}{2}\overline{\Delta}^{(2)}\right)^8}{2^7 \cdot 8!} \approx 5{,}69 \cdot 10^{-46}$$

También almacenamos en la memoria las siguientes constantes:

$$LN1 \simeq ln(2)_{hi} \qquad \text{(primeros 93 bits de } ln(2))$$
$$LN2 \simeq LN1 - ln(2)$$
$$LN3 \simeq \frac{1}{ln(2)}$$
$$\beta^{(1)} = \frac{1}{\Delta^{(1)}} = 256$$
$$\beta^{(2)} = \frac{1}{\Delta^{(2)}} = 2^{15}$$

Los valores $x^{(1)}$, $x^{(2)}$ obtenidos de la reducción de x, los expresamos como suma de dos variables

$$x^{(1)} = a^{(1)} + b^{(1)} \qquad x^{(2)} = a^{(2)} + b^{(2)}$$

Donde $a^{(1)}$ es la parte alta de $x^{(1)}$ y $b^{(1)}$ es la parte menos significativa de $x^{(1)}$. $a^{(2)}$ es la parte más significativa y $b^{(2)}$ es la parte menos significativa de $x^{(2)}$.

7.1. Algoritmo para calcular e^x

1. Calculamos n : $\quad n = [x \cdot LN3]$. Donde $[x]$ denota la función parte entera de x.
2. Calculamos $a^{(1)} = x - n \cdot LN1$, $\quad b^{(1)} = n \cdot LN2$ y expresamos $x^{(1)} = a^{(1)} + b^{(1)}$.
3. Calculamos i : $\quad i = \left[a^{(1)} \cdot \beta^{(1)}\right]$.
4. Buscamos $v_i^{(1)}, w_i^{(1)}, y_i^{(1)}$ en las tablas.
5. Calculamos $a^{(2)} = a^{(1)} - v_i^{(1)}$, $\quad b^{(2)} = b^{(1)} - w_i^{(1)}$ y expresamos $x^{(2)} = a^{(2)} + b^{(2)}$.
6. Calculamos j : $\quad j = \left[a^{(2)} \cdot \beta^{(2)}\right]$.
7. Buscamos $v_j^{(2)}, w_j^{(2)}, y_j^{(2)}$ en las tablas.
8. Calculamos $x^{(3)}$: $\quad x^{(3)} = a^{(2)} - v_j^{(2)} + b^{(2)} - w_j^{(2)}$.
9. Aproximamos $e^{x^{(3)}} - 1$: $\quad e^{x^{(3)}} - 1 \approx P(x^{(3)})$.
10. Calculamos z : $\quad z = 2^n y_i^{(1)} y_j^{(2)}$.
11. Calculamos c : $\quad c = z \cdot P(x^{(3)})$.
12. Calculamos y : $\quad y = z + c$.

Tenemos que $y \approx e^x$ con una precisión de 113 bits.

8. Funciones hiperbólicas

8.1. Seno hiperbólico y coseno hiperbólico

Las funciones seno hiperbólico y coseno hiperbólico están definidas en términos de la exponencial

$$\sinh(x) = \frac{e^x - e^{-x}}{2} \tag{8.1}$$

$$\cosh(x) = \frac{e^x + e^{-x}}{2} \tag{8.2}$$

Dado que la exponencial no tiene singularidades, $\sinh(x)$ y $\cosh(x)$ están definidas en todo el eje real y son positivas $\forall x > 0$. Éstas funciones tienen las siguientes propiedades de simetría:

$$\sinh(-x) = \frac{e^{-x} - e^x}{2} = -\frac{e^x - e^{-x}}{2} = -\sinh(x)$$
$$\cosh(-x) = \frac{e^{-x} + e^x}{2} = \cosh(x)$$

Para $x \geq 0$ se satisfacen las siguientes desigualdades

$$0 \leq \sinh(x) \leq \frac{1}{2}e^x \qquad 1 \leq \cosh(x) \leq e^x$$

Tenemos que $e^x \pm e^{-x} \simeq e^x$, $\forall x$ tal que $|x| > \lambda$ donde $\lambda \approx \frac{ln(2)}{2} \cdot (p+1) \Longrightarrow$ donde $p =$*número de bits significativos*. Por tanto, con cuádruple precisión $\lambda \approx 39{,}51..$ Si $|x| > \lambda$, entonces

$$|\sinh(x)| \simeq \cosh(x) \simeq \frac{1}{2}e^x \qquad [13]$$

Queremos calcular $\sinh(x)$ y $\cosh(x)$ para números de punto flotante con una representación en 128 bits. Por tanto, éstas funciones están definidas (no se produce desbordamiento) para x tal que

$$|x| \leq ln(2 \cdot FLT128\_MAX)$$

Ya que si $|x| > ln(2 \cdot FLT128\_MAX)$

$$\cosh(x) > \frac{e^{ln(2 \cdot FLT128\_MAX)}}{2} = \frac{2 \cdot FLT128\_MAX}{2} = FLT128\_MAX.$$

La definición de $\sinh(x)$ (8.1) no es eficiente para $|x|$ pequeño ya que la sustracción de cantidades casi iguales provoca una gran pérdida de precisión. Por lo tanto, utilizamos la función auxiliar $E(x) = e^x - 1$. Tenemos que

$$\begin{aligned}\sinh(x) &= \frac{e^x - e^{-x}}{2} = \frac{E(x) + 1 - \dfrac{1}{E(x)+1}}{2} = \frac{(E(x)+1)^2 - 1}{2(E(x)+1)} \\ &= \frac{E(x)^2 + 2E(x)}{2(E(x)+1)} = \frac{E(x)}{2}\frac{E(x)+2}{E(x)+1} = \frac{1}{2}\left(\frac{(E(x)+1-1)(E(x)+2)}{E(x)+1}\right) \\ &= \frac{1}{2}\left(\left(1 - \frac{1}{E(x)+1}\right)(E(x)+2)\right) = \frac{1}{2}\left(E(x)+2 - \frac{E(x)+2}{E(x)+1}\right) \\ &= \frac{1}{2}\left(E(x) + \frac{E(x)}{E(x)+1}\right)\end{aligned}$$

$$\begin{aligned}\cosh(x) &= \frac{e^x + e^{-x}}{2} = \frac{e^x}{2} + \frac{1}{2e^x} = \frac{e^{2x}+1}{2e^x} = \frac{e^{2x} + 1 - 2e^x + 2e^x}{2e^x} \\ &= 1 + \frac{e^{2x} - 2e^x + 1}{2e^x} = 1 + \frac{(e^x - 1)^2}{2e^x} = 1 + \frac{E(x)^2}{2e^x}\end{aligned}$$

8.1.1. Algoritmo para calcular sinh(x)

Dado un argumento x el siguiente algoritmo proporciona el cálculo de $\sinh(x)$ con una precisión de 113 bits.

1. Si $x = INF \quad o \quad NaN$, $\sinh(INF) = INF$ y $\sinh(NaN) = NaN$. Por tanto, retornamos $x \cdot x$ para generar un INF o un NaN, es decir, generamos un error.

2. Si $|x| < 2^{-57}$, $\sinh(x) = x + \frac{x^3}{6} + \frac{x^5}{120} + \cdots \Rightarrow \varepsilon_a(x) = |\sinh(x) - x| \Rightarrow$ $\varepsilon_a \approx \dfrac{(2^{-57})^3}{6} < 2^{-113}$. Por tanto, $\sinh(x) \approx x$ tiene una precisión cuádruple.

3. Por las propiedades de simetría de $\sinh(x)$ trabajaremos solo con valores positivos. Calculamos el signo de x y trabajamos con el valor absoluto de x. Calculamos $sig = sign(x), \quad y = |x|$

4. Si $2^{-57} \leq y < 40$, calculamos $\sinh(x) \simeq sig \cdot 0{,}5 \cdot \left(E(y) + \dfrac{E(y)}{E(y)+1}\right)$.

5. Si $40 \leq y \leq x_{max} \approx 11356{,}5234$, calculamos $\sinh(x) \simeq sig \cdot 0{,}5 \cdot e^y$.

6. Si $x_{max} < y \leq ln(2 \cdot FLT128\_MAX)$, la función $f(y) = e^y$ no está definida para $y > x_{max}$. Por lo tanto, tenemos que dividir x entre dos y calcular la exponencial. Calculamos

$$\sinh(x) \simeq sig \cdot 0{,}5 \cdot e^{\frac{y}{2}} \cdot e^{\frac{y}{2}}$$

7. Si $y > ln(2 \cdot FLT128\_MAX)$, se produce desbordamiento.

8.1.2. Algoritmo para calcular cosh(x)

Dado $x \in \mathbb{R}$, el siguiente algoritmo calcula $\cosh(x)$ con una precisión cuádruple.

1. Si $x = INF \quad o \quad NaN$, análogo a $\sinh(x)$ retornamos $x \cdot x$ para generar un error.
2. Si $|x| \leq 2^{-57}$, $\cosh(x) \approx 1$. Ya que $\cosh(x) = 1 + \frac{x^2}{2} + \frac{x^4}{24} + \cdots \Rightarrow$ $\varepsilon_a(x) = |\cosh(x) - 1| \approx \dfrac{\left(2^{-57}\right)^2}{2} = 2^{-113}$, es decir, $\cosh(x) \approx 1$ tiene una precisión de 113 bits.
3. Si $2^{-57} \leq x < \frac{ln(2)}{2}$, calculamos $\cosh(x) \simeq 1 + \dfrac{E(x)^2}{2e^x}$.
4. Si $\frac{ln(2)}{2} \leq x < 40$, calculamos

$$\cosh(x) \simeq 0{,}5 \cdot \left(e^x + \frac{1}{e^x}\right)$$

   Utilizamos la expresión $0{,}5 \cdot \left(e^x + \frac{1}{e^x}\right)$ para calcular $\cosh(x)$ en lugar de $0{,}5 \cdot \left(e^x + e^{-x}\right)$, ya que de la primera forma solo tenemos que calcular e^x y hacer una división. Así, evitamos calcular e^{-x}.
5. Si $40 \leq x \leq x_{max}$, calculamos $\cosh(x) \simeq 0{,}5 \cdot e^x$.
6. Si $x_{max} < x \leq ln(2 \cdot FLT128\_MAX)$, calculamos $\cosh(x) \simeq 0{,}5 \cdot e^{\frac{x}{2}} \cdot e^{\frac{x}{2}}$.
7. Si $x > ln(2 \cdot FLT128\_MAX)$, se produce desbordamiento.

8.2. Tangente hiperbólica

La tangente hiperbólica de x está definida como

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$\begin{aligned} \tanh(x) &= \frac{e^x - \dfrac{1}{e^x}}{e^x + \dfrac{1}{e^x}} = \frac{e^{2x} - 1}{e^{2x} + 1} = \frac{e^{2x} + 1 - 2}{e^{2x} + 1} \\ &= 1 - \frac{2}{e^{2x} + 1} \end{aligned} \tag{8.3}$$

La función tiene la siguiente propiedad de simetría

$$\tanh(-x) = \frac{e^{-x} - e^x}{e^{-x} + e^x} = -\frac{e^x - e^{-x}}{e^x + e^{-x}} = -\tanh(x)$$

Se cumple la siguiente desigualdad

$$-1 \leq \tanh(x) \leq 1 \qquad \forall x \in \mathbb{R}$$

La tangente hiperbólica está definida y es calculable (no se produce desbordamiento para ningún valor de x) para todo número de punto flotante sin ninguna restricción. Como hemos visto en el capítulo anterior $\forall x \in \mathbb{R}$ tal que $|x| > \lambda$ se cumple que $|\sinh(x)| \approx \cosh(x)$. Por lo tanto, si $|x| > \lambda$ $|\tanh(x)| \approx 1$, es decir, solo tenemos que calcular $\tanh(x)$ para $x \in [-40, 40]$.

Si $|x| < 1$ la definición (8.3) no es óptima, porque la cancelación hace que el cálculo sea inestable. Utilizamos la función $E(x)$.

$$\tanh(x) = \frac{\frac{1}{e^{-x}} - e^{-x}}{\frac{1}{e^{-x}} + e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}} = -\frac{E(-2x)}{E(-2x) + 2}$$

8.2.1. Algoritmo para tanh(x)

El siguiente algoritmo calcula $\tanh(x) \qquad \forall x \in \mathbb{R}$ con una precisión cuádruple

1. Si $x = INF \quad o \quad NaN$, igual que antes, retornamos $x \cdot x$ para generar un error.
2. Si $|x| < 2^{-57}$, $\tanh(x) = x - \frac{x^3}{3} + \cdots$, $\varepsilon_a(x) = |\tanh(x) - x| \approx \frac{(2^{-57})^3}{3} < 2^{-113}$. Entonces, $\tanh(x) \approx x$ con una precisión de 113 bits.
3. Calcular $sig = signbitq(x), \qquad x = fabs(x)$.
4. Si $2^{-57} \leq x < 1$, calculamos $\tanh(x) \simeq sig \cdot \left(-\frac{E(-2x)}{E(-2x) + 2}\right)$.
5. Si $1 \leq x \leq 40$, calculamos $\tanh(x) \simeq sig \cdot \left(1 - \frac{2}{E(2x) + 2}\right)$.
6. Si $x > 40$, calculamos $\tanh(x) = sig \cdot 1$.

9. Funciones auxiliares

En este capítulo, se explican algunas de las funciones auxiliares necesarias para la implementación de las funciones explicadas anteriormente.

frexp(x)

La función recibe un argumento x, y una variable exp. x es un número de punto flotante codificado en 128 bits

$$x = 2^E(1 + f) \qquad \text{con } E \text{ exponente sesgado}$$

Queremos expresar x de la forma

$$x = y \cdot 2^{exp} \quad \text{con } y \in \left[\frac{1}{2}, 1\right]$$

Primero separamos x en 2 partes:

-En la variable h_x guardamos los primeros 64 bits de x, es decir, la parte alta de x.

-En la variable L_x guardamos los últimos 64 bits de x, es decir, la cola de x.

Guardamos los primeros 16 bits de h_x en la variable *exp*. Estos 16 bits, corresponden al exponente sesgado de x. Calculamos

$$exp = exp - sesgo + 1 \Longrightarrow exp = exp - 16383 + 1$$

$E = e - sesgo$, entonces $exp = E + 1$. Por tanto

$$x = 2^{exp} \cdot 2^{-1}(1 + f)$$

Calculamos $y = 2^{-1}(1 + f)$ como $0 < f < 1 \Longrightarrow \frac{1}{2} < y < 1$

$$\boxed{x = 2^{exp} \cdot y}$$

scalbnq

La función $scalbnq(x, n)$ recibe 2 argumentos: x y n. Donde x es un número con una representación en punto flotante codificado en 128 bits, $x = 2^e(1 + f)$ y n es un entero. Queremos calcular $x \cdot 2^n$, pero no queremos introducir más error en las operaciones y puesto que multiplicar por 2^n solo es cambiar el exponente de x, la función *scalbnq* le suma n al exponente de x manipulando los bits del exponente.

Exponencial auxiliar $E(x) = e^x - 1$

Dado un argumento x, queremos calcular $e^x - 1$ con una precisión de 113 bits, para hacer dicho cálculo reducimos el argumento. La reducción del rango la llevamos a cabo separando el argumento en la forma

$$x = \ln(2)(k + r) = \ln(2)k + \ln(2)r$$

Donde $k \in \mathbb{Z}, |r| < \frac{1}{2}$. Definimos $F = \ln(2)r$. De esta forma

$$e^x = e^{\ln(2)k + F} = 2^k \cdot e^F$$

Sea $R_{7,8}(x)$ una aproximación *minimax* de $e^x - 1$ en el intervalo $\left[-\frac{\ln(2)}{2}, \frac{\ln(2)}{2}\right]$

$$R_{7,8}(x) = x + \frac{1}{2}x^2 + x^3\frac{P_7(x)}{Q_8(x)}, \qquad x \in \left[-\frac{\ln(2)}{2}, \frac{\ln(2)}{2}\right]$$

Donde $P_7(x)$ es un polinomio de grado 7 y $Q_8(x)$ es un polinomio mónico de grado 8 con los siguientes coeficientes:

- $p_0 = \quad 2{,}943520915569954073888921213330863757240 \cdot 10^8$
- $p_1 = -5{,}722847283900608941516165725053359168840 \cdot 10^7$
- $p_2 = \quad 8{,}944630806357575461578107295909719817253 \cdot 10^6$
- $p_3 = -7{,}212432713558031519943281748462837065308 \cdot 10^5$
- $p_4 = \quad 4{,}578962475841642634225390068461943438441 \cdot 10^4$
- $p_5 = -1{,}716772506388927649032068540558788106762 \cdot 10^3$
- $p_6 = 4{,}401308817383362136048032038528753151144 \quad \cdot 10^1$
- $p_7 = -4{,}888737542888633647784737721812546636240 \cdot 10^{-1}$
- $q_0 = \quad 1{,}766112549341972444333352727998584753865 \cdot 10^9$
- $q_1 = -7{,}848989743695296475743081255027098295771 \cdot 10^8$
- $q_2 = \quad 1{,}615869009634292424463780387327037251069 \cdot 10^8$
- $q_3 = -2{,}019684072836541751428967854947019415698 \cdot 10^7$
- $q_4 = \quad 1{,}682912729190313538934190635536631941751 \cdot 10^6$
- $q_5 = -9{,}615511549171441430850103489315371768998 \cdot 10^4$
- $q_6 = \quad 3{,}697714952261803935521187272204485251835 \cdot 10^3$
- $q_7 = -8{,}802340681794263968892934703309274564037 \cdot 10^1$
- $q_8 = \quad 1{,}000000000000000000000000000000000000000$

$$\left| e^x - 1 - x - \frac{1}{2}x^2 - x^3 \frac{P_7(x)}{Q_8(x)} \right| < 2^{-113}$$

$F \in \left[-\frac{\ln(2)}{2}, \frac{\ln(2)}{2} \right]$, entonces $R_{7,8}(F) = e^F - 1$. Por consiguiente

$$e^x - 1 = 2^k \cdot (R_{7,8}(F) + 1) - 1 \Rightarrow \boxed{e^x - 1 = 2^k \cdot R_{7,8}(F) + 2^k - 1}$$

10. Conclusiones

Las funciones: raíz cuadrada y cúbica, trigonométricas, exponencial e hiperbólicas estudiadas en este proyecto, solamente son una parte de las funciones implementadas en la librería *libquadmath*.

Se podría continuar analizando otro tipo de funciones de esta librería, sin embargo, una vez finalizado este trabajo vemos que el cálculo de casi todas las funciones analizadas, sigue la misma estrategia: primero reducir el argumento a un intervalo más pequeño cercano a 0, después aproximar la función en este nuevo intervalo utilizando diferentes técnicas como por ejemplo, aproximación por polinomios de Chebyshev, aproximación *minimax* o incluso utilizar tablas precalculadas y por último, recuperar el valor de la función a partir de los resultados anteriores aplicando propiedades matemáticas de la función en cuestión.

Para llevar a cabo este trabajo han sido necesarios conocimientos vistos durante la carrera en asignaturas como Elementos de Programación y Métodos Numéricos. En la realización del trabajo he adquirido otros nuevos como la aproximación por polinomios de Chebyshev o la aproximación *minimax*.

Una vez comprendidos los diferentes métodos matemáticos y técnicas a nivel teórico, la parte más costosa del trabajo ha sido entender el código de las funciones, es decir, comprender las implementaciones en C de estos métodos en las funciones. Ya que, por motivos de eficiencia los programas se acaban haciendo de forma no "matemáticamente natural".

Por otra parte, hemos visto que siempre se realizan los cálculos con números *hi* y *lo*, es decir, con la parte alta y la parte baja (la cola) del número. Para así, obtener mayor precisión en los cálculos y una precisión cuádruple en la evaluación de la función.

Hemos intentado razonar porque las funciones están programadas de esta manera atendiendo a resultados teóricos. Por lo tanto, una vez concluido el análisis, vemos que una posible linea de trabajo futura sería intentar mejorar las implementaciones de estas funciones, es decir, tratar de programar las funciones, basándonos en las técnicas estudiadas, para realizar los cálculos con una precisión cuádruple de una forma computacionalmente más rápida.

Referencias

[1] American National Standards Institute. IEEE Standard for Binary Floating Point Arithmetic: Aprobado 21 de marzo, 1985; Aprobado 26 de julio, 1985.

[2] Kahan, William. IEEE standard 754 for binary floating-point arithmetic. *Lecture Notes on the Status of IEEE,* 1996, vol. 754, no 94720-1776, p. 11.

[3] Taylor, John *Introduction to error analysis, the study of uncertainties in physical measurements.* University Science Books, 1997.

[4] Ezquerro Fernández, José A. *Iniciación a los métodos numéricos.* Universidad de la Rioja, 2012.

[5] Dahlquist, G; Björk, A. Traducido por N. Anderson, 1974. *Numerical Methods.* Prentice Hall, New Jersey, 1974.

[6] Barde, R. G.; Sherbert, D. R. *Introducción al análisis matemático de una variable.* Limusa (Noriega Editores), 1996.

[7] Ralston, A; Rabinowitz, P. *A first course in numerical analysis.* Courier Corporation, 2001.

[8] Maehly, H.; Witzgall, Ch. Tschebyscheff-Approximationen in kleinen Intervallen II. *Numerische Mathematik,* 1960, vol. 2, no 1, p. 293-307.

[9] Hart, John F. *Computer Appoximations.* Krieger Publishing Co., Inc. 1978.

[10] Tang, Ping Tak Peter. Table-lookup algorithms for elementary functions and their error analysis. En *Computer Arithmetic, 1991. Proceedings., 10th IEEE Symposium on.* IEEE, 1991. p. 232-236.

[11] Green, Robin. Faster math functions. En *Tutorial at Game Developers Conference.* 2002.

[12] Ziv, Abraham. Fast evaluation of elementary mathematical functions with correctly rounded last bit. *ACM Transactions on Mathematical Software (TOMS),* 1991, vol. 17, no 3, p. 410-423.

[13] Cody, W. J.; Waite, W. M. *Software manual for the elementary functions.* Prentice-Hall, 1980.