Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Métodos Numéricos

Trabajo Práctico Nº1

Errores en serie

Nombre	LU	Mail
Carla Livorno	424/08	carlalivorno@hotmail.com
Mariano De Sousa Bispo	389/08	marian_sabianaa@hotmail.com

Abstract

El siguiente trabajo consiste en analizar teóricamente la propagación de errores de cada una de las series propuestas, en función de la cantidad de dígitos de precisión de la aritmética utilizada para luego contrastar dicho análisis con los resultados empíricos. El trabajo pretende mostrar en los resultados lo que pueden producir pequeños errores de cálculo.

Aritmética finita Errores Pi

Índice

1. Introducción teórica

Cuando trabajamos con números reales en una computadora, nos encontramos con algunos factores limitantes para nuestros cálculos como lo son la aritmética finita y también el tipo de representación que se nos provee para trabajar.

Esto surge como una consecuencia de la necesidad de una memoria física finita de la computadora en contraste con la precisión infinita que requieren la mayoría de los números reales.

Luego, al intentar representar números en una computadora, cuya precisión va mas allá de la otorgada por la misma, surgen en la representación del número pequeños errores. Tambien se propagan los errores de redondeo y truncamiento debidos al almacenamiento a lo largo de las diferentes operaciones a que sometemos a los números. Estos errores, como muestra este trabajo, no son despreciables al realizarse ciertos tipos de cálculos.

Por otro lado, el álgebra de precisión finita es diferente al álgebra normal, la propiedad asociativa de los números reales no se cumple en general para los datos en punto flotante debido al error de redondeo. Si los valores a sumar son de muy diferente orden de magnitud se producirá un resultado más aproximado al correcto si se suman ordenadamente de menor a mayor. Esto nos demuestra que el orden en que se ejecutan las operaciones es importante para un computador.

2. Desarrollo

2.1. Analisis teórico

2.2. Serie de Gregory

Definimos primero el error de un término de la serie de Gregory.

Dado $n \in \mathbb{N}$, sea $\texttt{f(x,y)} = \frac{x}{y}$, donde $x = 1$ e $y = 2n + 1$.

El error de $\texttt{f}(x, y)$ es el error de un término de la serie de Gregory ya que x e y son de tipo entero (exactos).

Analicemos el error de $\texttt{f}(x, y)$:

$$\varepsilon_f(x, y, :) = \frac{\frac{x}{y}\varepsilon_x}{f} + \frac{\frac{-xy}{y^2}\varepsilon_y}{f} + \varepsilon_{(:)}^{(x,y)} = \frac{\frac{x}{y}\varepsilon_x}{\frac{x}{y}} + \frac{\frac{-x}{y}\varepsilon_y}{\frac{x}{y}} + \varepsilon_{(:)}^{(x,y)} =$$

$$\varepsilon_x - \varepsilon_y + \varepsilon_{(:)}^{(x,y)}$$

Teniendo esta información, queremos realizar el análisis teórico sobre la serie de Gregory para los primeros tres términos. Para poder realizarlo, llamamos a al primer término de la serie, b al segundo y c al tercero (llamamos $k = a + c$).

Sea $\texttt{g(a,c)} = a + c$

$$\varepsilon_g(a, c, +) = \frac{a}{a+c}\varepsilon_a + \frac{c}{a+c}\varepsilon_c + \varepsilon_{(+)}^{(a,c)}$$

Sea $\texttt{g'(k,b)} = k - b$

$\varepsilon_{g'}(k,b,-) = \frac{k}{k-b}\varepsilon_k - \frac{b}{k-b}\varepsilon_b + \varepsilon^{(k,b)}_{(-)}$

$\varepsilon_g(a+c,b,+) = \frac{a+c}{a+c-b}(\frac{a}{a+c}\varepsilon_a + \frac{c}{a+c}\varepsilon_c + \varepsilon^{(a,c)}_{(+)}) - \frac{b}{a+c-b}\varepsilon_b + \varepsilon^{(a+c,b)}_{(-)} =$

$\frac{a\varepsilon_a + c\varepsilon_c - b\varepsilon_b + (a+c)\varepsilon^{(a,c)}_{(+)}}{a-b+c} + \varepsilon^{(a+c,b)}_{(-)} = \varepsilon_{a+c-b} = \varepsilon_{\frac{\pi}{4}}$

Entonces el error total (ε_π) es el siguiente:

$\varepsilon_4 + \varepsilon_{a+c-b} + \varepsilon^{(4,a+c-b)}_{(*)} =$

$\varepsilon_4 + \frac{a\varepsilon_a + c\varepsilon_c - b\varepsilon_b + (a+c)\varepsilon^{(a,c)}_{(+)}}{a-b+c} + \varepsilon^{(a+c,b)}_{(-)} + \varepsilon^{(4,a+c-b)}_{(*)}$

Dado que $a = 1$, $b = \frac{1}{3}$ y $c = \frac{1}{5}$. Reemplazando por los términos correspondientes obtenemos:

$$\varepsilon_4 + \frac{\varepsilon_1 - \varepsilon_1 + \varepsilon^{(1,1)}_{(:)} - (\frac{1}{3})(\varepsilon_1 - \varepsilon_3 + \varepsilon^{(1,3)}_{(:)}) + (\frac{1}{5})(\varepsilon_1 - \varepsilon_5 + \varepsilon^{(1,5)}_{(:)}) + \frac{6}{5}\varepsilon^{(1,\frac{1}{5})}_{(+)}}{\frac{13}{15}} + \varepsilon^{(\frac{6}{5},\frac{1}{3})}_{(-)} + \varepsilon^{(4,\frac{13}{15})}_{(*)}$$

Si bien no conocemos cuál es el error exacto que posee cada operación, podemos acotarlas por el error máximo de todas ellas, llamémoslo k. Por lo tanto el resultado obtenido es menor igual a:

$k + \frac{k - (\frac{1}{3})k + (\frac{1}{5})k + \frac{6}{5}k}{\frac{13}{15}} + 2k = k + \frac{\frac{17}{15}k}{\frac{13}{15}} + 2k = k + \frac{17}{13}k + 2k = 3k + \frac{17}{13}k = \frac{56}{13}k < 5k$

Sabiendo que la implementación de estos algoritmos será realizada en una máquina con arítmetica binarias, podemos inferir que el error en los cálculos se producirá a partir de cierto bit, al que llamaremos t. El error entonces depende de la conversión de los valores enteros a números con coma, así como también

la presición que estemos utilizando. Tomamos de esta forma a $k = 2^{1-t}$.

Acotando el error de Gregory, tenemos que $\varepsilon_\pi < 5 * 2^{1-t}$

NOTA: Este último resultado (el valor de k), aplica tanto como al análisis previamente realizado, como a los dos subsiguientes (algoritmo de Machin y Ramanujan). Para simplificar los cálculos, y disminuir la probabilidad de arrastrar errores de cuentas, se acotará por k el que luego será reemplazado por 2^{1-t}.

2.3. Fórmula de Machin

Definimos primero el error de calcular $arctan(k)$ con $|k| < 1$. Para ello definimos el error de cada uno de los términos de la sumatoria.

Dado $n \in \mathbb{N}$, sea `f(x,y)` $= \frac{x}{y}$, donde $x = k^{2n+1}$ e $y = 2n + 1$.

Analicemos el error de `f`(x, y):

$$\varepsilon_f(x, y, :) = \frac{\frac{x}{y}\varepsilon_x}{f} + \frac{\frac{-xy}{y^2}\varepsilon_y}{f} + \varepsilon_{(:)}^{(x,y)} = \frac{\frac{x}{y}\varepsilon_x}{\frac{x}{y}} + \frac{\frac{-x}{y}\varepsilon_y}{\frac{x}{y}} + \varepsilon_{(:)}^{(x,y)} =$$

$$\varepsilon_x - \varepsilon_y + \varepsilon_{(:)}^{(x,y)} \quad (1)$$

Como y es de tipo entero (exacto) resta calcular el error de x.

Analicemos el error de $x = k^y$:

$$\varepsilon_x = \frac{yk^{y-1}k}{k^y}\varepsilon_k + \frac{k^y ln(k) y}{k^y}\varepsilon_y + \varepsilon_{(POT)}^{(k,y)} = y\varepsilon_k + ln(k)y\varepsilon_y + \varepsilon_{(POT)}^{(k,y)} \quad (2)$$

$\Rightarrow_{Reemplazando\ (2)\ en\ (1)}$

que el error de un termino de la arco tangente es:

$$y\varepsilon_k + ln(k)y\varepsilon_y + \varepsilon_{(POT)}^{(k,y)} - \varepsilon_y + \varepsilon_{(:)}^{(x,y)}$$

Teniendo esta información, queremos realizar el análisis teórico para los primeros tres términos de la sumatoria. Para poder realizarlo, llamamos a al primer término, b al segundo y c al tercero (llamamos $k = a + c$).

Sea `g(a,c)` $= a + c$

$\varepsilon_g(a, c, +) = \frac{a}{a+c}\varepsilon_a + \frac{c}{a+c}\varepsilon_c + \varepsilon_{(+)}^{(a,c)}$

Sea `g'(k,b)` $= k - b$

$\varepsilon_{g'}(k, b, -) = \frac{k}{k-b}\varepsilon_k - \frac{b}{k-b}\varepsilon_b + \varepsilon_{(-)}^{(k,b)}$

$\varepsilon_g(a + c, b, +) = \frac{a+c}{a+c-b}(\frac{a}{a+c}\varepsilon_a + \frac{c}{a+c}\varepsilon_c + \varepsilon_{(+)}^{(a,c)}) - \frac{b}{a+c-b}\varepsilon_b + \varepsilon_{(-)}^{(a+c,b)} =$

$\frac{a\varepsilon_a + c\varepsilon_c - b\varepsilon_b + (a+c)\varepsilon_{(+)}^{(a,c)}}{a-b+c} + \varepsilon_{(-)}^{(a+c,b)} = \varepsilon_{a+c-b}$

Para la fórmula de Machin se necesita calcular $arctan(\frac{1}{5})$ y $arctan(\frac{1}{239})$.

$\Rightarrow \varepsilon_{arctan}(\frac{1}{5}) =$

$$\frac{\frac{1}{5}(\varepsilon_{\frac{1}{5}} + ln(\frac{1}{5})\varepsilon_1 + \varepsilon_{(POT)}^{(\frac{1}{5},1)} - \varepsilon_1 + \varepsilon_{(:)}^{(\frac{1}{5},1)}) + (\frac{1}{5})^6(5\varepsilon_{\frac{1}{5}} + ln(\frac{1}{5})5\varepsilon_5 + \varepsilon_{(POT)}^{(\frac{1}{5},5)} - \varepsilon_5 + \varepsilon_{(:)}^{((\frac{1}{5})^5,5)})}{\frac{1}{5} + (\frac{1}{5})^5 - \frac{1}{3}(\frac{1}{5})^3} -$$

$$\frac{\frac{1}{3}(\frac{1}{5})^3(3\varepsilon_{\frac{1}{5}} + ln(\frac{1}{5})3\varepsilon_3 + \varepsilon_{(POT)}^{(\frac{1}{5},3)} - \varepsilon_3 + \varepsilon_{(:)}^{((\frac{1}{5})^3,3)}) + (\frac{1}{5} + (\frac{1}{5})^6)\varepsilon_{(+)}^{(\frac{1}{5},(\frac{1}{5})^6)}}{\frac{1}{5} + (\frac{1}{5})^6 - \frac{1}{5}(\frac{1}{5})^3} + \varepsilon_{(-)}^{(\frac{1}{5}+(\frac{1}{5})^6, \frac{1}{3}(\frac{1}{5})^3)}$$

$\Rightarrow \varepsilon_{arctan}(\frac{1}{239}) =$

$$\frac{\frac{1}{239}(\varepsilon_{\frac{1}{239}} + ln(\frac{1}{239})\varepsilon_1 + \varepsilon_{(POT)}^{(\frac{1}{239},1)} - \varepsilon_1 + \varepsilon_{(:)}^{(\frac{1}{239},1)}) + \frac{(\frac{1}{239})^5}{5}(5\varepsilon_{\frac{1}{239}} + ln(\frac{1}{239})5\varepsilon_5 + \varepsilon_{(POT)}^{(\frac{1}{239},5)} - \varepsilon_5 + \varepsilon_{(:)}^{((\frac{1}{239})^5,5)})}{\frac{1}{239} + \frac{1}{5}(\frac{1}{239})^5 - \frac{1}{3}(\frac{1}{239})^3} -$$

$$\frac{\frac{(\frac{1}{239})^3}{3}(3\varepsilon_{\frac{1}{239}} + ln(\frac{1}{239})3\varepsilon_{\frac{1}{239}} + \varepsilon_{(POT)}^{(\frac{1}{239},3)} - \varepsilon_3 + \varepsilon_{(:)}^{((\frac{1}{239})^3,3)}) + (\frac{1}{239} + \frac{(\frac{1}{239})^5}{5})\varepsilon_{(+)}^{(\frac{1}{239}, \frac{(\frac{1}{239})^5}{5})}}{\frac{1}{239} + \frac{1}{5}((\frac{1}{239})^5 - \frac{1}{3}(\frac{1}{239})^3} + \varepsilon_{(-)}^{(\frac{1}{239}+\frac{(\frac{1}{239})^5}{5}, (\frac{1}{239})^3)}$$

Finalmente calculamos el error de la fórmula de Machin.

Sea `f'` $= z * 4$,

$\varepsilon_{f'}(z, 4, *) = \varepsilon_z + \varepsilon_4 + \varepsilon_{(*)}^{(z,4)}$ (3)

Por otro lado calculamos el error de z instanciando g' con $k = 4 * arctan(\frac{1}{5})$ y $b = arctan(\frac{1}{239})$

$\varepsilon_z = \frac{4*arctan(\frac{1}{5})}{4*arctan(\frac{1}{5})-arctan(\frac{1}{239})}\varepsilon_{4*arctan(\frac{1}{5})} - \frac{arctan(\frac{1}{239})}{4*arctan(\frac{1}{5})-arctan(\frac{1}{239})}\varepsilon_{arctan(\frac{1}{239})} + \varepsilon_{(-)}^{(\frac{1}{5},\frac{1}{239})}$(4)

$\Rightarrow$ *Reemplazando* (4) *en* (3)

$\frac{4*arctan(\frac{1}{5})}{4*arctan(\frac{1}{5})-arctan(\frac{1}{239})}\varepsilon_{4*arctan(\frac{1}{5})} - \frac{arctan(\frac{1}{239})}{4*arctan(\frac{1}{5})-arctan(\frac{1}{239})}\varepsilon_{arctan(\frac{1}{239})} + \varepsilon_{(-)}^{(\frac{1}{5},\frac{1}{239}))} + \varepsilon_4 + \varepsilon_{(*)}^{(4*arctan(\frac{1}{5}-arctan(\frac{1}{239})),4)}$

$\Rightarrow$

$\frac{4*arctan(\frac{1}{5})}{4*arctan(\frac{1}{5})-arctan(\frac{1}{239})}(\varepsilon_{arctan(\frac{1}{5})} + \varepsilon_4 + \varepsilon_{(*)}^{(arctan(\frac{1}{5}),4)}) - \frac{arctan(\frac{1}{239})}{4*arctan(\frac{1}{5})-arctan(\frac{1}{239})}\varepsilon_{arctan(\frac{1}{239})} + \varepsilon_{(-)}^{(\frac{1}{5},\frac{1}{239}))} + \varepsilon_4 + \varepsilon_{(*)}^{(z,4)} <$

$\Rightarrow$ *Acotando los errores por* k, *es menor a*:

$< \frac{4*arctan(\frac{1}{5})}{4*arctan(\frac{1}{5})-arctan(\frac{1}{239})}3k - \frac{arctan(\frac{1}{239})}{4*arctan(\frac{1}{5})-arctan(\frac{1}{239})}k + 3k$

$\Rightarrow$

$< \frac{4*arctan(\frac{1}{5})}{4*arctan(\frac{1}{5})-arctan(\frac{1}{239})}3k + \frac{arctan(\frac{1}{239})}{4*arctan(\frac{1}{5})-arctan(\frac{1}{239})}3k + 3k$

$\Rightarrow$

$< (\frac{4*arctan(\frac{1}{5})+arctan(\frac{1}{239})}{4*arctan(\frac{1}{5})-arctan(\frac{1}{239})} + 1)3k < (\frac{5*arctan(\frac{1}{5})}{3*arctan(\frac{1}{239})} + 1)3k < 240k$

Reemplazando $k = 2^{1-t}$ nos queda que el error relativo del

algoritmo de Machin:

$\varepsilon_\pi < 240 * 2^{1-t}$

2.4. Serie de Ramanujan

Definimos primero el error de un término de la serie de Ramanujan.

Dado $n \in \mathbb{N}$, sea `f(x,y)` $= \frac{x}{y}$, donde $x = (4n)!(1103+26390n)$ e $y = (n!)^4 396^{4n}$.

Analicemos el error de `f`(x, y):

$$\varepsilon_f(x, y, :) = \frac{\frac{x}{y}\varepsilon_x}{f} + \frac{\frac{-xy}{y^2}\varepsilon_y}{f} + \varepsilon_{(:)}^{(x,y)} = \frac{\frac{x}{y}\varepsilon_x}{\frac{x}{y}} + \frac{\frac{-x}{y}\varepsilon_y}{\frac{x}{y}} + \varepsilon_{(:)}^{(x,y)} =$$

$$\varepsilon_x - \varepsilon_y + \varepsilon_{(:)}^{(x,y)} \quad (1)$$

Como calculamos x e y en punto flotante tenemos que calcular el error cometido.
Tanto para el cálculo del error de x como de y necesitamos calcular el error del producto.

Sea `f'(x',y')` $= x' * y'$ entonces,

$$\varepsilon_{f'}(x', y', *) = \varepsilon_{x'} + \varepsilon_{y'} + \varepsilon_{(*)}^{(x',y')} \quad (2)$$

Analicemos el error de $x = (4n)!(1103 + 26390n)$:

Instanciando $x' = (4n)!$ e $y' = 1103 + 26390n$ en (2) obtenemos:

$$\varepsilon_x = \varepsilon_{(4n)!} + \varepsilon_{1103+26390n} + \varepsilon_{(*)}^{((4n)!,1103+26390n)}$$

Resta calcular el error de x' ya que y' se calcula en enteros

(exacto). Basandonos en nuestra implementacion podemos decir que $x' = x_1 * (4n) * (4n-1) * (4n-2) * (4n-3)$ donde x_1 es una variable de punto flotante que acumula el resultado del factorial obtenido en el calculo del termino anterior.

$$\varepsilon_{x'} = \varepsilon_{x_1} + \varepsilon_{4n} + \varepsilon_{(*)}^{(x_1,4n)} + \varepsilon_{4n-1} + \varepsilon_{(*)}^{(x_1*4n,4n-1)} + \varepsilon_{4n-2} + \varepsilon_{(*)}^{(x_1*4n*(4n-1),4n-2)} + \varepsilon_{4n-3} + \varepsilon_{(*)}^{(x_1*4n*(4n-1)*(4n-2),4n-3)} \quad (3)$$

$\Rightarrow_{Reemplazando\ (3)\ en\ (2)}$

$$\varepsilon_x = \varepsilon_{x_1} + \varepsilon_{4n} + \varepsilon_{(*)}^{(x_1,4n)} + \varepsilon_{4n-1} + \varepsilon_{(*)}^{(x_1*4n,4n-1)} + \varepsilon_{4n-2} + \varepsilon_{(*)}^{(x_1*4n*(4n-1),4n-2)} + \varepsilon_{4n-3} + \varepsilon_{(*)}^{(x_1*4n*(4n-1)*(4n-2),4n-3)} + \varepsilon_{y'} + \varepsilon_{(*)}^{((4n)!,(1103+26390n))}$$

Analicemos el error de $y = (n!)^4 396^{4n}$:

Instanciando $x' = (n!)^4$ e $y' = 396^{4n}$ en (2) obtenemos:

$$\varepsilon_y = \varepsilon_{(n!)^4} + \varepsilon_{396^{4n}} + \varepsilon_{(*)}^{((n!)^4,396^{4n})}$$

Resta calcular el error de x' e y'.

Para calcular el error de x' llamamos $k_1 = n!$ y $k_2 = 4$.

Veamos entonces el error de $x' = k_1^{k_2}$

$$\varepsilon_{x'} = \frac{k_1^{k_2-1} k_2 k_1}{k_1^{k_2}} \varepsilon_{k_1} + \frac{k_2 k_1^{k_2} ln(k_1)}{k_1^{k_2}} \varepsilon_{k_2}$$

$$\varepsilon_{x'} = k_2 \varepsilon_{k_1} + k_2 ln(k_1) \varepsilon_{k_2} \quad (4)$$

Basandonos nuevamente en nuestra implementación podemos

decir que $k_1 = f'(k'_1, n)$ donde k'_1 es una variable de punto flotante que acumula el resultado del factorial obtenido en el calculo del termino anterior.

$$\varepsilon_{k_1} = \varepsilon_{k'_1} + \varepsilon_n + \varepsilon_{(*)}^{(k'_1,n)}$$

$$\Rightarrow_{Reemplazandoen(4)} \varepsilon_{x'} = k_2\varepsilon_{k'_1} + \varepsilon_n + \varepsilon_{(*)}^{(k'_1,n)} + k_2 ln(k_1)\varepsilon_{k_2}$$

$$\Rightarrow \varepsilon_{x'} = 4\varepsilon_{k'_1} + \varepsilon_n + \varepsilon_{(*)}^{(k'_1,n)} + 4ln(n!)\varepsilon_4$$

Instanciando en (4) $k_1 = 396$ y $k_2 = 4n$ (4n es calculado en enteros)

$$\varepsilon_{y'} = 4n\varepsilon_{396} + 4nln(396)\varepsilon_{4n}$$

Teniendo el error de x' e y' podemos obtener el error de y reemplazando dichos errores en (2)

$$\varepsilon_y = 4\varepsilon_{k'_1} + \varepsilon_n + \varepsilon_{(*)}^{(k'_1,n)} + 4ln(n!)\varepsilon_4 + 4n\varepsilon_{396} + 4nln(396)\varepsilon_{4n} + \varepsilon_{(*)}^{((n!)^4,396^4 n)}$$

Finalmente, con el error de x e y obtenemos reemplazando en (1) el error de calcular un término de la serie de *Ramanujan*.

$$\varepsilon_{x_1}+\varepsilon_{4n}+\varepsilon_{(*)}^{(x_1,4n)}+\varepsilon_{4n-1}+\varepsilon_{(*)}^{(x_1*4n,4n-1)}+\varepsilon_{4n-2}+\varepsilon_{(*)}^{(x_1*4n*(4n-1),4n-2)}+\varepsilon_{4n-3}+\varepsilon_{(*)}^{(x_1*4n*(4n-1)*(4n-2),4n-3)}+\varepsilon_{y'}+\varepsilon_{(*)}^{((4n)!,(1103+26390n))}-4\varepsilon_{k'_1}+\varepsilon_n+\varepsilon_{(*)}^{(k'_1,n)}+4ln(n!)\varepsilon_4+4n\varepsilon_{396}+4nln(396)\varepsilon_{4n}+\varepsilon_{(*)}^{((n!)^4,396^4 n)}+\varepsilon_{(:)}^{((4n)!(1103+26390n),(n!)^4 396^{4n})}$$

Queremos realizar el análisis teórico sobre la serie de Ramanujan para los primeros tres términos de la sumatoria. Para poder realizarlo, llamamos a al primer término de la serie, b al segundo y c al tercero (llamamos $k = a + b$).

Sea `g(k,c)` $= k + c$

$\varepsilon_g(k, c, +) = \frac{k}{k+c}\varepsilon_k + \frac{c}{k+c}\varepsilon_c + \varepsilon_{(+)}^{(k,c)}$

$\varepsilon_g(a+b, c, +) = \frac{a+b}{a+b+c}(\frac{a}{a+b}\varepsilon_a + \frac{b}{a+b}\varepsilon_b + \varepsilon_{(+)}^{(a,b)}) + \frac{c}{a+b+c}\varepsilon_c + \varepsilon_{(+)}^{(a+b,c)} =$

$= \frac{a\varepsilon_a + b\varepsilon_b + c\varepsilon_c + (a+b)\varepsilon_{(+)}^{(a,b)}}{a+b+c} + \varepsilon_{(+)}^{(a+b,c)}$

Dado $a = 1103$, $b = \frac{659832}{396^4}$ y $c = \frac{8!*53883}{2^4*396^8}$ el error de los tres primeros términos es el siguiente:

El error de a es el error de representar 1103 ya que lo adicionamos al final sin calcularlo.

$\varepsilon_b = \varepsilon_1 + \varepsilon_4 + \varepsilon_{(*)}^{(1,4)} + \varepsilon_3 + \varepsilon_{(*)}^{(4,3)} + \varepsilon_2 + \varepsilon_{(*)}^{(12,2)} + \varepsilon_1 + \varepsilon_{(*)}^{(24,1)} + \varepsilon_{1103+26390} + \varepsilon_{(*)}^{(24,1103+26390)} - 4\varepsilon_1 + \varepsilon_1 + \varepsilon_{(*)}^{(1,1)} + 4\varepsilon_{396} + 4ln(396)\varepsilon_4 + \varepsilon_{(*)}^{(1,396^4)} + \varepsilon_{(:)}^{(24(1103+26390),396^4)}$

$\Rightarrow$ *Acotando todos los errrores por* k, *y usando que* $ln(396)<6$ *es menor a*: $39k$

$\varepsilon_c = \varepsilon_{24} + \varepsilon_8 + \varepsilon_{(*)}^{(24,8)} + \varepsilon_7 + \varepsilon_{(*)}^{(192,7)} + \varepsilon_6 + \varepsilon_{(*)}^{(1344,6)} + \varepsilon_5 + \varepsilon_{(*)}^{(8064,5)} +$

$\varepsilon_{1103+52780} + \varepsilon_{(*)}^{(8!,(1103+52780))} - 4\varepsilon_1 + \varepsilon_2 + \varepsilon_{(*)}^{(1,2)} + 4ln(2)\varepsilon_4 + 8\varepsilon_{396} + 8ln(396)\varepsilon_8 + \varepsilon_{(*)}^{(16,396^8)} + \varepsilon_{(:)}^{(8!(1103+52780),16*396^8)}$

$\Rightarrow$ *Acotando todos los errrores por k, y usando que* $ln(396)<6$ *es menor a:* $71k$

$$\frac{1103\varepsilon_{1103}+\frac{659832}{396^4}\varepsilon_b+\frac{8!*53883}{2^4*396^8}\varepsilon_c+(1103+\frac{659832}{396^4})\varepsilon_{(+)}^{(1103,\frac{659832}{396^4})}}{1103+\frac{659832}{396^4}+\frac{8!*53883}{2^4*396^8}}+\varepsilon_{(+)}^{(1103+\frac{659832}{396^4},\frac{8!*53883}{2^4*396^8})}$$

Por otro lado calculamos el error de `h(z)` $= \sqrt{z}$.

$$\varepsilon_h(z,\surd) = \frac{\frac{1}{2\sqrt{z}}}{\sqrt{z}} z\varepsilon_z + \varepsilon_{(\surd)}^{(z)} = \frac{1}{2\sqrt{z}}\frac{1}{\sqrt{z}} z\varepsilon_z + \varepsilon_{(\surd)}^{(z)} = \frac{\varepsilon_z}{2} + \varepsilon_{(\surd)}^{(z)}$$

Conociendo el error de $\mathtt{f}(x,y)$ calculamos el error de $\frac{\sqrt{8}}{9801}$ instanciando $x = \sqrt{z}$ e $y = 9801$.

$$\Rightarrow \frac{\varepsilon_z}{2} + \varepsilon_{(\surd)}^{(z)} - \varepsilon_{9801} + \varepsilon_{(:)}^{(\sqrt{8},9801)}$$

Sea `h'(x,y)` $= x*y$, donde $x = \frac{\sqrt{8}}{9801}$ e $y = \sum_{n=0}^{\infty} \frac{(4n)!(1103+26390n)}{(n!)^4 396^{4n}}$.

sabemos que: $\varepsilon_{h'}(x,y,*) = \varepsilon_x + \varepsilon_y + \varepsilon_{(*)}^{(x,y)}$

$$\Rightarrow \varepsilon_{h'}(x,y,*) = \varepsilon_{\frac{\sqrt{8}}{9801}} + \varepsilon_{\sum_{n=0}^{2} \frac{(4n)!(1103+26390n)}{(n!)^4 396^{4n}}} + \varepsilon_{(*)}^{(\frac{\sqrt{8}}{9801},\sum_{n=0}^{2} \frac{(4n)!(1103+26390n)}{(n!)^4 396^{4n}})} =$$

$$\frac{\varepsilon_z}{2} + \varepsilon_{(\surd)}^{(z)} - \varepsilon_{9801} + \varepsilon_{(:)}^{(\sqrt{8},9801)} + \frac{1103\varepsilon_{1103}+\frac{659832}{396^4}\varepsilon_b+\frac{8!*53883}{2^4*396^8}\varepsilon_c+(1103+\frac{659832}{396^4})\varepsilon_{(+)}^{(1103,\frac{659832}{396^4})}}{1103+\frac{659832}{396^4}+\frac{8!*53883}{2^4*396^8}} + \varepsilon_{(+)}^{(1103+\frac{659832}{396^4},\frac{8!*53883}{2^4*396^8})} + \varepsilon_{(*)}^{(\frac{\sqrt{8}}{9801},\sum_{n=0}^{2} \frac{(4n)!(1103+26390n)}{(n!)^4 396^{4n}})}$$

Conociendo el error de $\mathtt{f}(x,y)$ calculamos el error de $\frac{1}{\frac{\sqrt{8}}{9801}\sum_{n=0}^{2} \frac{(4n)!(1103+26390n)}{(n!)^4 396^{4n}}}$

instanciando $x = 1$ e $y = \frac{\sqrt{8}}{9801} \sum_{n=0}^{2} \frac{(4n)!(1103+26390n)}{(n!)^4 396^{4n}}$.

$$\Rightarrow \varepsilon_\pi = \varepsilon_1 - \frac{1103\varepsilon_{1103} + \frac{659832}{396^4}\varepsilon_b + \frac{8!*53883}{2^4*396^8}\varepsilon_c + (1103+\frac{659832}{396^4})\varepsilon_{(+)}^{(1103,\frac{659832}{396^4})}}{1103+\frac{659832}{396^4}+\frac{8!*53883}{2^4*396^8}} + \varepsilon_{(+)}^{(1103+\frac{659832}{396^4},\frac{8!*53883}{2^4*396^8})} + \varepsilon_{(:)}^{(1,\sum_{n=0}^{2}\frac{(4n)!(1103+26390n)}{(n!)^4396^{4n}})}$$

$\Rightarrow_{Tomando\ k\ el\ maximo\ error}$

$k - \frac{2207k}{1103+\frac{659832}{396^4}+\frac{8!*53883}{2^4*396^8}} + 2k < k - \frac{2207k}{1103+0+0} + 2k = k - \frac{2207k}{1103} + 2k$
$< k - \frac{5k}{2} + 2k < k - 3k + 2k < k + 3k + 2k = 6k$

Tomamos de esta forma a $k = 2^{1-t}$.

Acotando el error de Ramanujan, tenemos que $\varepsilon_\pi < 6*2^{1-t} = 3 * 2^{2-t}$

2.5. Implementación

Al necesitar manejar presición arbitraria los tipos de datos nativos del lenguaje C++ no satisfacían nuestras necesidades.

Una solución encontrada a este problema, fue implementar un tipo de datos (clase *Real*) con esta funcionalidad (presición variable). La clase *Real* implementa los operadores básicos: suma, resta, multiplicación. división y asignación, los observadores sobre la presición y truncamiento, y funciones para visualización de la información del *Real* en stdout. Se exporta además dos métodos con funcionalidad importante, uno de ellos para convertir el *Real* en *double* (tipo nativo de C++) y otro método que dado un *double* refresca esta información en la instancia *Real* correspondiente.

Como precondición de este trabajo práctico la presición máxima de digitos (en base 2) del cálculo de π debe ser a lo sumo 51 bits. Sabiendo que C++ cumple con el standar IEEE-754, el cual establece 52 bits de mantisa para un tipo de datos *double*, utilizamos a este como cimiento de nuestra clase *Real*. Las operaciones aritméticas sobre *Real* consisten en convertirlos en *double*, realizar la operación correspondiente y transformar el resultado nuevamente

en un *Real*, aplicándole previamente el algoritmo de truncamiento acorde a los parámetros de entrada del programa (cantidad de dígitos).

Fuera de la clase implementamos el cálculo de raíz cuadrada, arco tangente y exponenciación de *Real*. Se tomó esta decisión ya que consideramos que estas no son operaciones básicas, por lo que el uso de estas sólo tiene sentido en instancias de *Real* en situaciones complejas (por ejemplo el cálculo de π). Tomamos como modelo la implementación de estas operaciones en C++ que requiere la inclusión de una biblioteca (cmath).

La implementación de todos los algoritmos del cálculo de π maximizan el uso de variables de tipo entero ya que las cuentas en número flotante poseen error mientras que enteros no. Cuando los cálculos dejan de tener sentido en el mundo de los enteros o su presición no es suficiente, se crea una instancia de *Real* que represente al valor entero y se continúa operando sobre este tipo. Por ese motivo una instancia de *Real* se puede generar sólo a partir de un entero de 64 bits con signo.

A continuación detallamos las optimizaciones de los algoritmos:

- Gregory: Refactorizamos nuestra implementación original de la serie que consistía en respetar el orden de los sumandos según el enunciado como las operaciones internas de cada uno de ellos. A continuación se detalla en lenguaje matemático los cambios realizados.

  $$\sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = \sum_{n\in\mathbb{N}/n\ mod(2)=0}^{\infty} \frac{1}{2n+1} - \sum_{n\in\mathbb{N}/n\ mod(2)=1}^{\infty} \frac{1}{2n+1}$$

  Al hacerlo de esta manera en la ejecución de cada iteración del algoritmo evitamos el cálculo de una potencia al costo de almacenar dos números que acumulan las sumas.

  Por otro lado, decidimos calcular $2n + 1$ en un tipo de dato entero de 64 bits sin signo ya que de esta forma podemos evitar el error en el cálculo (a diferencia de si se realizara con números flotantes) y sería necesario un n muy grande (cantidad de sumandos de la serie) para producir $overflow$.

- Machin: Este algoritmo ejecuta la función *arctan* la cual realiza la optimización antes mencionada.

- Ramanujan: Este algoritmo fue rediseñado en varias oportunidades al observar diferentes problemas en cada implementación que realizamos. Entre sus operaciones encontramos la función factorial la cual en pocas iteraciones (en enteros sin signo de 64 bits) produce $overflow$. Sabiendo que el rango de valores representable en *double*, y por consiguiente en *Real* es mayor al de cualquier entero de 64 bits, optamos por realizar la operación factorial en nuestro tipo de datos (*Real*).

    - NOTA: No olvidar que la cantidad de valores representable en *double* y enteros de 64 bits es la misma, ya que las permutaciones binarias posibles de ambos es igual a $2^{64} - 1$.

Nuestra primer aproximación fue definir la función factorial, la cual dado un n calculaba la $\prod_{k=1}^{n} k$. Observamos que la cantidad de operaciones elementales que factorial realiza en cada llamada es lineal en función de n. El uso que se le da en el algoritmo de *Ramanujan* es calcular el factorial de números consecutivos pudiéndose entonces, reutilizar el resultado de la iteración previa.

A continuación se muestra un breve pseudcódigo del uso de la función factorial en el algoritmo de *Ramanujan* (primera implementación).

```
Ramanujan(n)
    for j=0  to n
        factorial(4 * j)
              .
              .
              .
    end
```

Sabiendo que factorial es O(n) y la cantidad de iteraciones del ciclo for es n y suponiendo que el resto de las operaciones del ciclo son

constantes el algoritmo tiene complejidad cuadrática en función de n.

A continuación se muestra un breve pseudocódigo del cálculo del factorial acumulando en una variable el resultado de la iteración anterior.

```
Ramanujan(n)
    acum_fact = 1
    for j=1  to n
        i = 4 * j
        acum_fact = acum_fact * (i - 1) * (i - 2) * (i - 3) * i
                .
                .
                .
    end
```

De esta forma, se realiza una cantidad constante de multiplicaciones en cada iteración del ciclo for para el cálculo del factorial (suponiendo el resto de las operaciones del ciclo de costo constante) el algoritmo tiene complejidad lineal en función de n.

La complejidad final del algoritmo de *Ramanujan* no es esta ya que entre las operaciones del ciclo for se encuentra la operación potencia con costo lineal en función del exponente $(4 * j)$. La complejidad es de $O(n^2)$.

Cabe mencionar que el primer término de la serie es un valor constante y entero, por lo que decidimos excluirlo del cálculo de la sumatoria y adicionarlo al final (por ese motivo la implementación final itera de 1 a n).

Para facilitar el desarrollo del programa se generaron varias funciones auxiliares de impresión de información. A la hora de hacer los test no era suficiente la visualización provista por la salida standard de C++. Estas

visualizaciones como por ejemplo mostrar el *double* bit a bit facilitaron la comprensión y analisis de errores programáticos.

3. Resultados

Para analizar las diferencias y/o similitudes en la aplicación práctica de los tres algoritmos implementados se generaron gráficos que pasan a detallarse a continuación.

El primero de ellos consiste en gráficar el error relativo en función de la cantidad de iteraciones para cada uno de los algoritmos. Se utilizó precisión fija de 51 bits ya que por precondición la cantidad de dígitos de la mantisa debe ser menor a 52, es decir, con 51 dígitos minimizamos el error. Conseguimos la precisión deseada mediante truncamiento. El eje del gráfico que corresponde al *error relativo* está en escala logarítmica para poder apreciar mejor los valores correspondientes dado que estos decrecen exponencialmente.

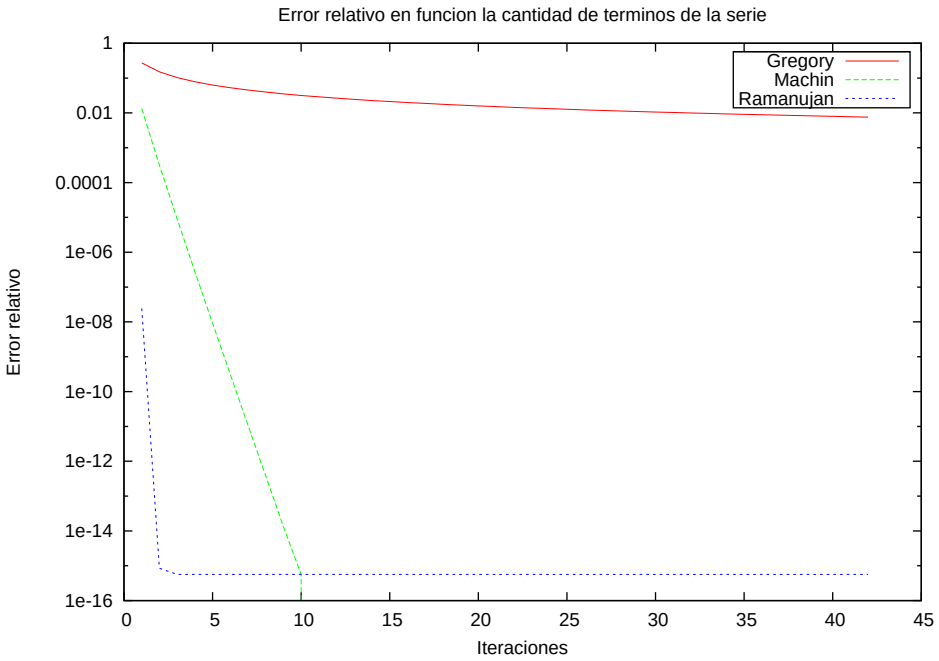

Figura 1: Comparación del error relativo de las tres series variando de 1 a 42 la cantidad de términos calculados con presición de 51 bits en la mantisa.

Para un posterior análisis sobre la información aquí suministrada, se agrega a continuación, un gráfico, detalle del anterior, se mostrará el error relativo entre la iteración 3 y 12 para el algoritmo de *Machin* y el de *Ramanujan* con los mismos parámetros (51 digitos de presición).

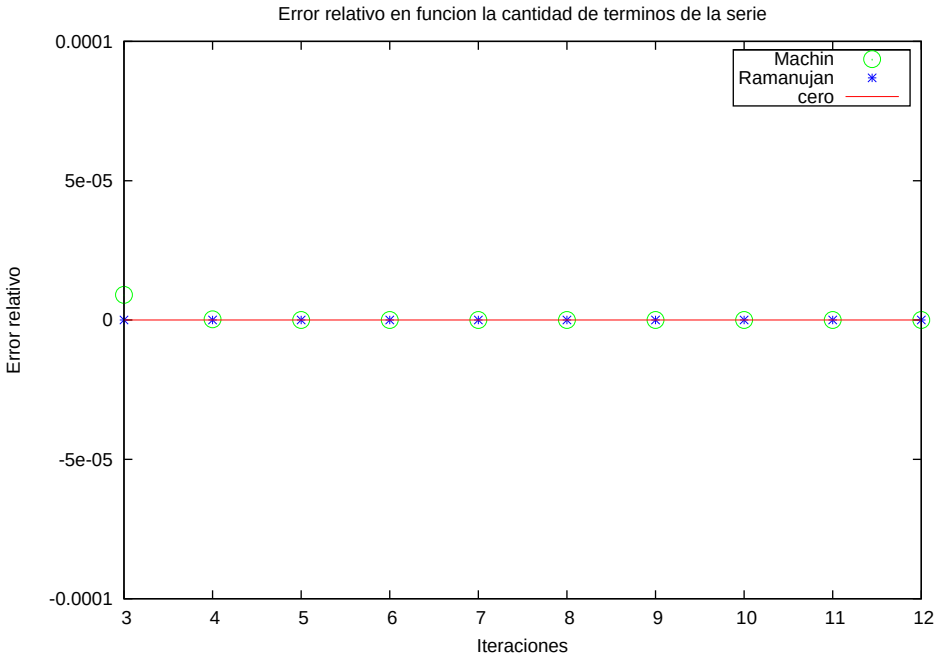

Figura 2: Comparación del error relativo la serie de Machin y de Ramanujan variando de 3 y 12 la cantidad de términos calculados con presición de 51 bits en la mantisa.

Se realizaron además, experimentaciones sobre el error relativo en función de la cantidad de bits de precisión (variando este parámetro de 1 a 51). A continuación se detallan los resultados de estos experimientos.

Se corrieron las pruebas hasta 42 iteraciones debido a que la serie de *Ramanujan* es informativa hasta esa iteración (a partir de 43 iteraciones devuelve *nan*), es decir, el error relativo presentado corresponde al error cometido al calcular 42 términos de la serie.

El gráfico se presenta bajo una escala logarítmica en y.

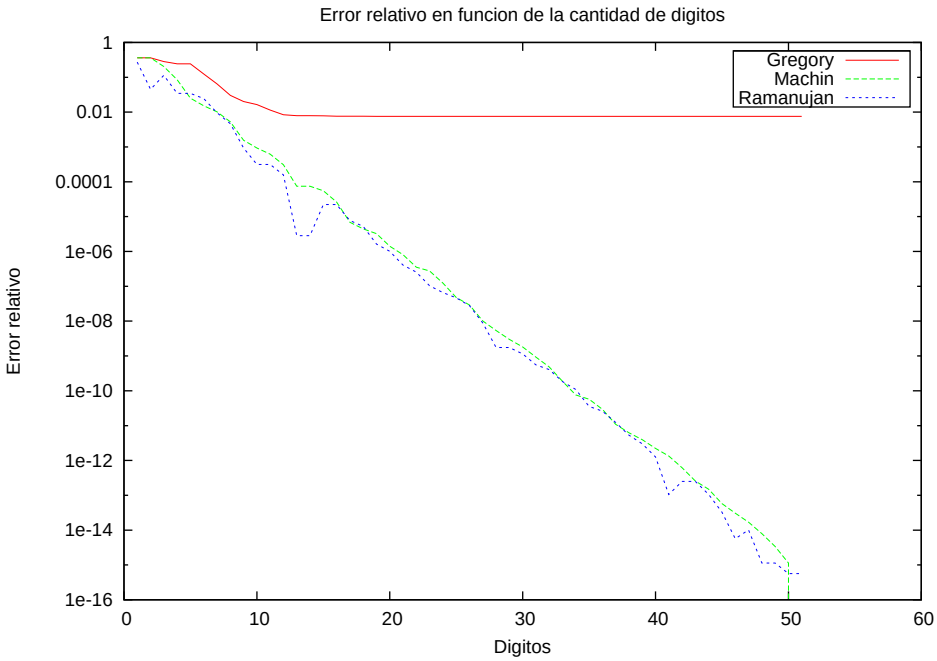

Figura 3: Comparación del error relativo de las tres series en el término 42 variando de 1 y 51 la cantidad de bits en la mantisa.

Consideramos que es pertinente mostrar un nuevo gráfico fijando la cantidad de iteraciones en un valor menor para verificar si existe alguna diferencia o no. Elegimos arbitrariamente correr las pruebas con 5 iteraciones.

Figura 4: Comparación del error relativo de las tres series en el término 5 variando de 1 y 51 la cantidad de bits en la mantisa.

NOTA:

Para hacer más simple el análisis de los gráficos, cada par de valores correspondientes a puntos consecutivos del eje x, se los unió mediante una recta, funcionalidad utilizada de GNUPLOT.

No se utilizaron polinomios interpoladores para aproximar por la curva o recta que pase por todos los puntos correspondiente a una misma fuente de datos, estas conclusiones fueron sacadas utilizando experimentación empírica, suponiendo que el comportamiento cuando los valores del eje x tienden a infinito se corresponden a los de los valores graficados.

El último gráfico corresponde al error relativo en función de la cantidad de dígitos, pero evaluando las cotas obtenidas en el análisis teórico con el error relativo de implementación.

La información obtenida en este gráfico, nos ayudará a poder discernir cuál es la relación que existe entre estos dos mundos.

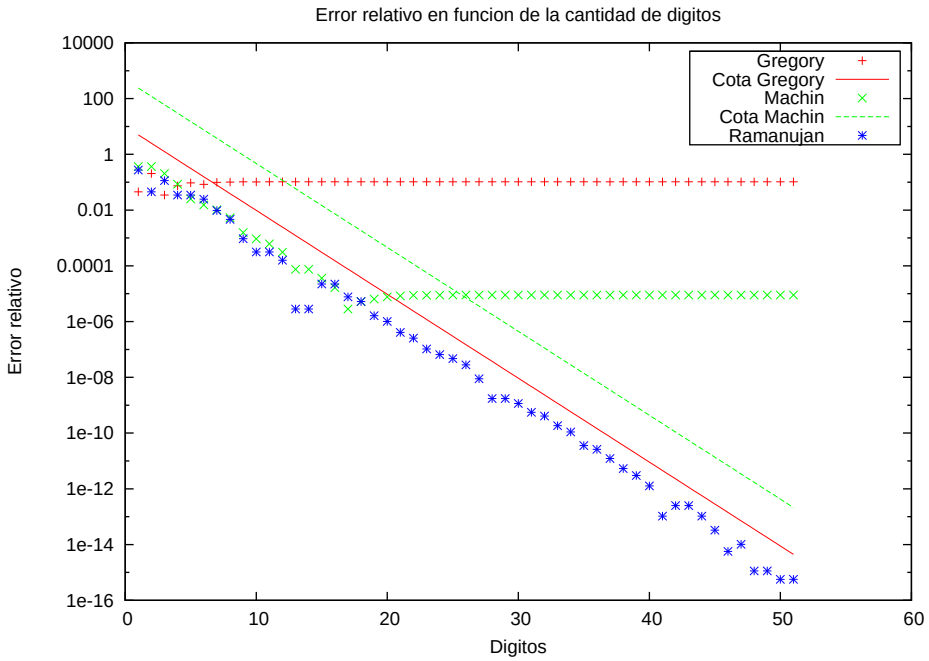

Figura 5: Comparación del error relativo de las tres series y los errores teóricos en función de la cantidad de bits de mantisa.

4. Discusión

En esta sección, buscaremos conclusiones a la información suministrada por los gráficos de la sección anterior.

El primer gráfico presenta una primer aproximación al error relativo de los tres algoritmos (Figura:1). Observamos que a medida que la cantidad de términos de las series aumenta, el error cometido en el cálculo de π de cada una de ellas disminuye.

El error introducido al aproximar π con la serie de $Gregory$ es logarítmico (en la escala del gráfico) en función de la cantidad de términos de la serie calculados. Podemos realizar un cambio de escala en el eje y, transformándolo en lineal obtenemos que el error de la serie de $Greory$ decrece linealmente conforme aumenta la cantidad de iteraciones.

En la primer aproximación a la comprensión del error relativo cometido al aproximar π con la fórmula de $Machin$ en función de la cantidad de iteraciones, vemos que los valores obtenidos en cada iteración forman una recta con pendiente negativa. Como la escala utilizada para representar dicho error es la logarítmica, podemos decir que el error decrece exponencialmente.

Además observamos que a partir de la décima iteración el gráfico no muestra de forma clara lo que ocurre con el error en la fórmula de $Machin$. Suponemos que se debe al hecho de que los valores que representan a $Machin$ se superponen con los valores correspondientes a $Ramanujan$.

Por otro lado, podemos apreciar que con pocas iteraciones la serie de $Ramanujan$ obtiene un error constante, similar observación puede hacerse a $Machin$ a partir de la iteración diez.

Generamos entonces un nuevo gráfico (Figura:2) para darle sustento a estas conclusiones, o refutarlas y buscar un nuevo enfoque sobre los resultados anteriores.

Con este gráfico nos vemos tentados a validar lo supuesto en un principio, ya que se puede apreciar que entre estas iteraciones el error cometido por ambos métodos es el mismo. Más aún se podría concluir a partir del gráfico que el error es cero lo cual no es cierto. Al ser un error tan chico el gráfico no ayuda a la correcta interpretación de los datos. Sabemos que, a pesar de seguir iterando no se podria obtener una mejor aproximación a π ya que el impedimento viene dado por la cantidad de digitos de presición utilizada. A partir de esto, deducimos que al tratarse de magnitudes tan pequeñas los gráficos son informativos sólo en rasgos generales y no en detalles.

Como tercer resultado, tenemos al gráfico que muestra el error relativo con respecto a la cantidad de dígitos, fijando la cantidad de iteraciones en 42 (Figura:3)

La apreciación (lineal/exponencial) del decrecimiento de los errores en función de la cantidad de dígitos es la misma que en función de la cantidad de iteraciones para *Ramanujan* y *Machin*. El primero de los dos en este caso, decrece lineal (pensándolo en escala logarítmica) sin cambiar el valor de su pendiente a partir de cierto valor. Parece reducir entonces, de manera exponencial el error cometido conforme aumenta la presición utilizada.

Gregory muestra una pendiente logarítmica monótona decreciente hasta un t cercano a doce, donde según este gráfico, se estabiliza y se transforma en constante. Si esto sucede, significaría que no tiene sentido seguir calculando más iteraciones, ya que el error de estos dígitos no van a mejorar con respecto al valor exacto de la constante a calcular, es decir, si al variar la cantidad de bits el error no disminuye, entonces existe al menos un bit con diferencia (entre *Gregory* y el exacto), la cual nunca será cero (la diferencia, vala la redundancia). Por lo tanto, podríamos suponer que la serie converge a π más el error relativo, lo que implicaría que Gregory en efecto no calcula dicha constante. Esta serie de pasos lógicos, nos hace llevar a la conclusión que la escala del gráfico nos hace perder información sensible y crucial para el entendimiento, ya que en caso contrario, estaríamos ante la presencia de un absurdo.

Observamos al igual que en el gráfico anterior que el error cometido por *Machin* con 51 dígitos de presición no aparece graficado mientras que con el resto de las precisiones si. Concluimos a partir de esto, que a pesar que el error cometido por ambas series (*Machin* y *Ramanujan*) es similar con 42 iteraciones, se necesita la máxima presicion brindada para que se haga totalmente despreciable la diferencia, si existe.

En la Figura:4, plantea fijar la cantidad de iteraciones en un valor menor (cinco en este caso), y analizar si existe diferencias con respecto al anterior de 42.

Vemos que con poca presición los errores siguen siendo similares.

Machin consigue mejores aproximaciones que *Gregory* fijando una cantidad de iteraciones, cuando la presición es baja pierde mayor información al truncar lo que se ve reflejado en la diferencia de errores al aumentar dicha presición (necesita más dígitos para estabilizarse).

En el primer y segundo algoritmo, *Gregory* y *Machin*, podemos inferir que existe una relación entre la cantidad de iteraciones de la serie la precisión utilizada, ahora al tener una cantidad inferior de iteraciones, el hecho que exista la posibilidad de calcular con precisión mayor a 30 bits (para *Machin* por ejemplo) no aporta. Esto en cambio, no sucede tan notoriamente con *Ramanujan*. Si bien los tres algoritmos necesitan de realizar la sumatoria infinita para converger a la constante, dado un valor fijo para el cálculo de la sumatoria y/o bits fijos, su mejor aproximación vendrá dada por *Ramanujan*, seguido de *Machin* y luego *Gregory*.

En el último gráfico adjuntado, Figura:5 vemos que un cierto una cierta cantidad de dígitos utilizada el análisis teórico ya no sirve como cota. Esto sucede cuando los errores cometidos se 'estabilizan' (a pesar de tener más bits de presición para operar el error disminuye imperceptiblemente). Pensamos que el hecho de que en un momento la cota queda por debajo de los valores se debe a que restringimos a *tres* la cantidad de términos de las series y la cota esta expresada en función a la cantidad de dígitos utilizados. Por ese motivo podemos ver que *Ramanujan* se mantiene siempre por debajo de la cota, es decir, lo hace porque a medida que trabaja con una mayor presición logra disminuir su error.

5. Conclusiones

El cálculo sobre aritmética finita presenta un gran desafío para los desarrolladores, al intentar proponer nuevas maneras que minimicen esta pérdida de información, como para quienes utilizan estas herramienta para el cálculo. Si bien hasta el momento no hemos encontrado herramientas que puedan presentar un resultado exacto en todos los casos, conocer como acotar estos errores, nos da una ayuda extra para poder entender a qué corresponde los resultados en la experimentación.

Inclusive, sabiendo el error cometido en los cálculos, la representación de esta información tanto en gráficos, como en cualquier medio observable de datos, también agrega un margen de error que hay que ser muy precavido que existe, para no llegar a falsas conclusiones.

Existe un factor dominante a la hora de implementar algoritmos o utilizarlos, que es el requerimiento de eficiencia. Los tres algoritmos muestran esta estrecha relación que hay entre precisión y eficiencia. Si bien el tercer algoritmo es el que más rápido aproxima, también su complejidad asíntotica es la mayor (se podría demostrar que es $O(n^2)$), al igual que *Machin* y complejidad lineal para *Gregory*. Dependiendo del contexto en el que se utilice, quizás la pérdida de precisión, se justifica por el orden de magnitud de diferencia en el cálculo.

Deberíamos tener en cuenta también, que utilizamos como base, los tipos nativos de C++, si necesitamos por ejemplo, mayor precisión, agregamos un nuevo factor que es la cantidad de memoria utilizada, la cual en principio podría no ser acotada.

Encontrar el balance entre estos tres problemas reales es muy complejo, pero herramientas como el cálculo del error relativo, nos ayudan a bislumbrar con mejor claridad, cuál es la mejor opción para cada contexto.

Nos han surgido varias implementaciones alternativas, así como desiciones estructurales quizás mejores, equilibrando el tiempo disponible y la complejidad de llevarlas acabo, optamos por esta clase *Real*. Sin embargo, nos gustaría denotar algunas de esas ideas a continuación.

Con respecto a la implementación de los algoritmos, existen funciones que podrían haber sido optimizadas, como el cálculo de la potencia el cual podría haberse implementado de forma mas inteligente utilizando programación dinámica, recordando los valores ya calculado para no repetirlos.

Podría haberse evitado la generación de un objeto *Real* por cada iteración de los algoritmos de aproximación a π.

Entre alguna de las ideas tratadas en el grupo, surgió lamentablemente cuando el trabajo ya estaba avanzado, la idea de trabajar con fracciones y no reales. Es decir, si consiguiéramos una representacion sin errores para el numerador y denominador (ejemplo, enteros de precisión arbitraria), podríamos realizar todas las cuentas evitando casi toda pérdida de presición (salvo la representación quizás por salida estándar). Si bien, nos interesó muchísimo esta idea, su implementación hubiera sido mucho más compleja, al no poder contar con los algoritmos ya existentes en *doubles*.

La clase *Real* se implementó de forma tal que acepte la realización de operaciones entre objetos con distinta precisión con el objeto de agregar a los resultados diferentes tests como por ejemplo, analizar el impacto de utilizar en el denominador presición mayor a la del numerador, etc. Se implementó además de forma tal que deje al usuario elegir entre truncamiento o redondeo a t bits. Encontramos ciertos casos donde el redondeo no se porta de la forma esperada, y no pudimos corregirlo. Nos parece adecuado igual, dejarlo implementado, con el objeto tanto de presentar el concepto general del programa generado, como no producir quizás imperceptibles errores que compliquen la entrega del trabajo.

6. Apéndices

6.1. Apéndice A: Enunciado

Laboratorio de Métodos Numéricos - Primer cuatrimestre 2011
Trabajo Práctico Número 1: Errores en serie ...

El objetivo del trabajo práctico es analizar el comportamiento numérico de varios métodos para aproximar π cuando cada método se implementa por medio de una aritmética finita de t dígitos. Los métodos consisten en evaluar las siguientes series hasta un cierto término:

Serie de Gregory (1671):

$$\frac{\pi}{4} = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$$

Fórmula de Machin (1706):

$$\frac{\pi}{4} = 4\arctan(1/5) - \arctan(1/239), \qquad \text{con } \arctan(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{2n+1} \quad \text{cuando } |x| < 1$$

Serie de Ramanujan (1914):

$$\frac{1}{\pi} = \frac{\sqrt{8}}{9801} \sum_{n=0}^{\infty} \frac{(4n)!\,(1103 + 26390n)}{(n!)^4\, 396^{4n}}$$

El trabajo práctico consta de dos partes:

1. **Análisis teórico**

   Analizar la propagación de errores en el tercer término (en función de los errores de los dos primeros) de cada una de las series propuestas, en función de la precisión aritmética utilizada. Emplear estos resultados en la argumentación de sus conclusiones luego de realizar el análisis empírico del punto 2.

2. **Análisis empírico**

   Implementar los tres métodos con aritmética binaria de punto flotante con t dígitos de precisión en la mantisa (el valor t debe ser un parámetro de la implementación, $t < 52$) y comparar los errores relativos de los resultados obtenidos en los tres casos y con las cotas de error del punto anterior. Realizar al menos los siguientes experimentos numéricos:

   a) Reportar el error relativo de cada método en función de la cantidad de dígitos t de precisión en la mantisa, para cantidades fijas de términos n de la serie.

   b) Reportar el error relativo de cada método en función de la cantidad de términos n de la serie correspondiente, para cantidad de dígitos t de precisión.

   c) Comparar los resultados obtenidos en *a*) y *b*) con las cotas de error calculadas en el punto 1.

   d) (Opcional) Explorar distintas formas de implementar las fórmulas (realizar las cuentas) de cada cada método.

   Se deben presentar los resultados de estas pruebas en un formato conveniente para su visualización y análisis. Sobre la base de los resultados obtenidos, ¿se pueden extraer conclusiones sobre la conveniencia de utilizar uno u otro método?

6.2. Apéndice B: Código fuente

6.3. PI: Modo de compilación

Para compilar se hace uso de la herramienta Makefile.

Abrir una terminal dentro la carpeta *code* entregada y escribir el comando "make", pulsar enter.

NOTA:

PI fue programado en plataforma Linux.

6.4. PI: Modo de uso

Una vez compilado escriba en la terminal (posicinado sobre la misma ruta en la que lo compiló) $./pi\ --help$ para recibir ayuda detalla sobre como utilizarlo o $./pi$ para un resumen de la misma.

Recibe al menos dos parámetros para funcionar, el primero el cual selecciona cuál será el modo de uso y el segundo qué algoritmos utilizará. Llamándolo con estos únicos dos parámetros, mostrará los valores por defecto utilizados para ser los cálculos y realizará la ejecución, siendo estos modificables ingresándolos como parámetros de entrada.

A continuación se escribe el esquema sobre la utilización del programa, recordar que los primeros dos valores encerrados entre corchetes son obligatorios:
'./pi [modoUso] [metodo] [tdigitos] [cantidadIteraciones] [trunca?]'

Por defecto el programa imprime los resultados por la salida estándar, si se desea por ejemplo, guardar los resultados obtenidos en un archivo de texto puede escribir la siguiente sentencia.

'./pi [modoUso] [metodo] [tdigitos] [cantidadIteraciones] [trunca?] >> [ruta/nombreArchivo]'
NOTA:

El operador >> utilizado en la consola para redigir la salida estándar a un archivo, es un programa propio de Linux, no siendo este implementado por nosotros.

7. Referencias

Para la realización de los gráficos, se tomó de la siguiente página, el valor de los primeros dígitos de π

http://www.super-computing.org/

Esta página Web, pertenece al laboratorio 'Kanada', el cual pertenece al centro de computación de la Universidad de Tokio.