Trabajo Práctico Nº 1

Sin margen de error (numérico)

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

Primer Cuatrimestre - 2007
Grupo X Completar:mandar mail publicar grupo y esperar respuesta

Guillermo Túnez, Matías Albanesi, Mercedes Bianchi

9 de abril de 2007

Nombre	LU	E-mail
Guillermo Túnez	790/00	gtunez2002@yahoo.com.ar
Matías Albanesi Cáceres	522/00	mac@sion.com
Completar!!!	Completar!!!	Completar!!!

Extracto: Completar Lunes!!!
Palabras Clave: Completar Lunes!!!

Índice

1. Introducción

Matías: yo opino que acá deberían estar al menos mencionados cualquier concepto teórico que aparezca en el informe - salvo los triviales. No vamos a explicar la suma, por ejemplo.

1.1. Aritmética finita y Análisis de error

Opinión: Creo que habría que hacer que esta sección tenga más sentido, coherencia reagrupando los párrafos por ejemplo

Cuando se usa una calculadora o computadora digital para realizar cálculos numéricos, se debe considerar un error inevitable, el llamado error de redondeo o truncamiento.

Truncamiento: Este consiste en usar solamente las n primeras cifras de un número para representarlo.

Redondeo: La representación del número tendrá una cantidad fija de dígitos, solo que ahora se utilizará el número de n dígitos mas cercano al valor real.

Este error se origina porque la aritmética realizada en una máquina involucra números con solo un número finito de dígitos, con el resultado de que muchos cálculos se realizan con representaciones aproximadas de los números verdaderos. En una computadora, sólo un subconjunto relativamente pequeño del sistema de los números reales se usa para representar a todos los números reales. Este subconjunto contiene sólo números racionales, positivos y negativos, y almacena una parte fraccionaria, llamada mantisa, junto con una parte exponencial, llamado exponente. Aunque muchas veces la representación de un numero en la máquina es exacta, esta después de haber sufrido alguna operación aritmética pierde su precision total, es decir que hay un error de representación.

El objeto de este trabajo es realizar un análisis empírico del comportamiento de la función $G(x)$ para distintos valores de x, calculada con aritmética de punto flotante de precision finita.

1.2. Errores de representación y operaciones matemáticas

Los números reales representados en punto flotante son los más utilizados en computación científica. Como hemos observado al normalizar un número real se comete un error. Incluso números con una representación finita en decimal, como 0.1, o 27.9, tienen una representación binaria infinita, tienen decimales binarios periódicos. Por ello, cuando estos números se almacenan

en punto flotante se debe "cortar" este número a una cantidad finita de bits, y se incurre en un error de representación flotante. Estos errores, de truncado o redondeo, son inevitables. Además, cuando realizamos operaciones elementales como sumar o multiplicar números también se incurre en un error adicional, que es similar al de normalización.

1.3. Ejemplo: Cancelación catastrófica

Hemos observado en este trabajo práctico que al sumar (restar) números de diferente (igual) signo, y de módulo(magnitud creo que es mas adecuado que modulo!!!) muy parecido, se produce una pérdida de dígitos significativos en el resultado, incluso cuando éste es exacto. Este fenómeno se denomina cancelación. En principio, la cancelación no es algo malo. Sin embargo, si el resultado de ésta se utiliza en operaciones sucesivas, la pérdida de dígitos significativos puede generar la propagación de errores que pueden reducir fuertemente la exactitud del resultado final, y se produce una cancelación catastrófica.

2. Análisis del problema

Como etapa previa al desarrollo en computadora de los cálculos pedidos en el enunciado, analicemos las propiedades de las funciones a calcular.

Recordemos la definición de estas funciones:

$$\begin{aligned} T(x) &= \begin{cases} 1 & \text{si } x = 0 \\ \frac{e^x - 1}{x} & \text{en caso contrario} \end{cases} \\ Q(x) &= \left| x - \sqrt{x^2 + 1} \right| - \frac{1}{x + \sqrt{x^2 + 1}} \\ G(x) &= T(Q(x)^2) \end{aligned}$$

Pese a su aparente dificultad, se prueba que $Q(x) = 0 \ \forall x \in \mathbb{R}$ (ver más abajo). Por lo tanto,

$$G(x) = T(Q(x)^2) = T(0^2) = T(0) = 1$$

por definición de $T(x)$.

Notemos que $T(x)$ es una función partida, definida en 1 para $x = 0$. Esta definición permite que $T(x)$ sea continua en $x = 0$ y en todo $\mathbb{R}$, como se demuestra más abajo.

¿Que implica esto? Interpretamos que el espíritu del problema no es obtener el valor más cercano posible al teórico, porque éste es constante. Se trata, en cambio, de observar las características de distintas operaciones de punto flotante de la máquina y los errores de cálculo que producen.

Esto nos permitió tomar distintas decisiones de implementación, privilegiando facilitad de desarrollo por encima de la minimización de errores en los cálculos.

2.1. Demostración de $Q(x) = 0 \ \forall x \in \mathbb{R}$

Queremos probar que, matemáticamente, $Q(x) = 0$ para cualquier valor $x \in \mathbb{R}$

$$Q(x) = \left| x - \sqrt{x^2 + 1} \right| - \frac{1}{x + \sqrt{x^2 + 1}}$$

Para esto separaremos la demostración para distintos valores de x.

- Para $x < 0$:

$$\begin{aligned}\left|x - \sqrt{x^2+1}\right| &= \left| - |x| - \sqrt{x^2+1}\right| \\ &= \sqrt{x^2+1} + |x|\end{aligned}$$

Entonces:

$$\begin{aligned}Q(x) &= \left|x - \sqrt{x^2+1}\right| - \frac{1}{x + \sqrt{x^2+1}} \\ &= \left(\sqrt{x^2+1} + |x|\right) - \frac{1}{-|x| + \sqrt{x^2+1}} \\ &= \frac{\left(\sqrt{x^2+1} + |x|\right)\left(\sqrt{x^2+1} - |x|\right) - 1}{\left(\sqrt{x^2+1} - |x|\right)} \\ &= \frac{\sqrt{x^2+1}^2 - |x|^2 - 1}{\left(\sqrt{x^2+1} - |x|\right)} \\ &= \frac{x^2 + 1 - x^2 - 1}{\left(\sqrt{x^2+1} - |x|\right)} \\ &= 0\end{aligned}$$

- Para $x \geq 0$, veamos que $x - \sqrt{x^2+1} \leq 0$:

  - Para $0 \leq x \leq 1$:

$$\begin{aligned}0 &\leq x \\ 0 &\leq x^2 \\ 1 &\leq x^2 + 1 \\ 1 &\leq \sqrt{x^2+1} \\ x &\leq \sqrt{x^2+1} \\ x - \sqrt{x^2+1} &\leq 0\end{aligned}$$

- Y para $1 \leq x$:

$$\begin{aligned} 1 &\leq x \\ x &\leq x^2 \\ x^2 &< x^2+1 \\ x &< \sqrt{x^2+1} \\ x-\sqrt{x^2+1} &< 0 \end{aligned}$$

Por lo tanto para $x \geq 0$ tenemos:

$$\left|x-\sqrt{x^2+1}\right| = \sqrt{x^2+1}-x$$

Entonces:

$$\begin{aligned} Q(x) &= \left|x-\sqrt{x^2+1}\right| - \frac{1}{x+\sqrt{x^2+1}} \\ &= \left(\sqrt{x^2+1}-x\right) - \frac{1}{\sqrt{x^2+1}+x} \\ &= \frac{\left(\sqrt{x^2+1}-x\right)\left(\sqrt{x^2+1}+x\right)-1}{\left(\sqrt{x^2+1}+x\right)} \\ &= \frac{\sqrt{x^2+1}^2-x^2-1}{\left(\sqrt{x^2+1}+x\right)} \\ &= 0 \end{aligned}$$

2.2. Demostración de continuidad de $T(x)$

Queremos ver que $T(x)$ es continua en $x = 0$. Para eso tenemos que probar:

1. $\lim_{x\to 0^+} T(x) = \lim_{x\to 0^-} T(x) = L (L \in \mathbb{R})$,
2. $T(0) = L$

En conclusión

$$\lim_{x\to 0} T(x) = T(0)$$

Si calculamos el

$$\lim_{x\to 0}\left(\frac{e^x-1}{x}\right)$$

nos quedaria una indefinición del tipo $(\frac{0}{0})$.

Según la Regla de L'Hopital si tenemos dos funciones $f(x)$ y $g(x) \neq 0$, son derivables y $\lim_{x\to c}\frac{f(x)}{g(x)} = \frac{0}{0}$ o $\frac{\infty}{\infty}$, entonces cuando $\lim_{x\to c}\frac{f'(x)}{g'(x)}$ existe o es ∞.

En este caso la función $f(x) = e^x - 1$ y $g(x) = x$. Estas dos funciones son derivables y tanto el $\lim_{x\to 0^+}\frac{f(x)}{g(x)} = \frac{0}{0}$ como el $\lim_{x\to 0^-}\frac{f(x)}{g(x)} = \frac{0}{0}$.

Entonces estamos dentro de las condiciones de la Regla de L'Hopital y por lo tanto:

1. $$\lim_{x\to 0^+}\frac{f(x)}{g(x)} = \lim_{x\to 0^+}\frac{f'(x)}{g'(x)} = \lim_{x\to 0^+}\frac{(e^x-1)'}{(x)'} = \lim_{x\to 0^+}\frac{e^x}{1} = 1$$

2. $$\lim_{x\to 0^-}\frac{f(x)}{g(x)} = \lim_{x\to 0^-}\frac{f'(x)}{g'(x)} = \lim_{x\to 0^-}\frac{(e^x-1)'}{(x)'} = \lim_{x\to 0^-}\frac{e^x}{1} = 1$$

Por otro lado sabemos, por definición, que $f(0) = 1$.

Entonces $T(x)$ es continua en $x = 0$ ya que :

1. $\lim_{x\to 0^+} T(x) = \lim_{x\to 0^-} T(x) = 1 (1 \in \mathbb{R})$, y
2. $T(0) = 1$.

Fuera de $x = 0$, T(x) es continua en todo $\mathbb{R}$ por ser cociente de funciones continuas.

3. Análisis previo al desarrollo en computadora

La primer decisión a tomar consiste en plataforma y compilador a utilizar. Como el desarrollo debe cumplir con el requisito de poder compilarse y ejecutarse con el software disponible en los laboratorios del Departamento de Computación, optamos por computadoras con procesadores compatibles con la arquitectura Intel x86, y el compilador g++ de la Free Software Foundation.

Luego analizamos las distintas alternativas posibles para implementar la aritmética de punto flotante de precisión variable, que consisten en:

- Desarrollar un formato propio, junto con una implementación de las operaciones sobre dicho formato.
- Utilizar la aritmética de punto flotante que provee la computadora y realizar ajustes para reducir la precisión de los resultados.

Nos decidimos rápidamente por la segunda opción ya que desarrollar un nuevo formato e implementación insumiría más tiempo de desarrollo y prueba, y el problema al cual se aplica no amerita el esfuerzo. Se trata de encontrar errores más que de evitarlos con números de precisiones mayores.

Al usar la aritmética de punto flotante que provee el hardware, podemos elegir entre cualquiera de los tres formatos: **float**, **double** o **long double**. Optamos por este último ya que poseía mayor cantidad de bits en la mantisa y tendríamos un rango de precisiones mayor para elegir; también encontramos que puede accederse fácilmente a la mantisa por separado del exponente y signo lo que simplifica la implementación.

Al operar con formatos nativos del equipo, se realizarán las operaciones y luego de cada una deberá ajustarse el resultado por alguno de los siguiente métodos:

- redondeo
- truncamiento

Pese a que el redondeo es preferible ya que el error relativo de representación es menor (la mitad) que el error cometido al realizar truncamiento, optamos por truncar el resultado ya que la implementación sería muy simple. Y debido a que estamos buscando resultados inexactos, al operar con aritmética de punto flotante y truncar tendremos errores más evidentes. Cabe mencionar que al realizar un truncamiento luego de cada operación de cálculo estamos agregando error a cada paso y obtendremos un resultado final con elevado alto grado de error.

3.1. El formato en punto flotante IEEE-754

Un formato de punto flotante es una estructura de datos que especifica los campos que abarcan un número de punto flotante, la disposición de esos campos, y su interpretación aritmética. Un formato de almacenamiento de punto flotante especifica cómo un número de punto flotante se almacena en memoria. El estándar de IEEE define los formatos, pero deja a los implementadores la opción de las formas de almacenamiento.

Existen varios formatos para la representación de números flotantes en la computadora, aunque el estándar es el formato ANSI/IEEE standard 754-1985, que llamaremos IEEE-754 para abreviar.

Existen 2 formatos básicos de representación de punto flotantes: single y double. También presenta dos clases extendidas de representación de números flotantes: single extendida y double extendida. El estándar no prescribe la precisión y el tamaño exactos de estos formatos, sino que especifica la precisión y el tamaño mínimos.

Una implementación concreta de este estándar es el formato de punto flotante extendido de la arquitectura Intel x86, de 80 bits de longitud, que dedica 64 bits para la mantisa - este formato se denomina **long double** en el lenguaje C/C++.

3.2. El Formato de precisión doble extendido de Intel x86

El formato *double* extendido de la arquitectura Intel x86, denominado **long double** en el lenguaje C/C++, cumple con las condiciones del formato doble extendido IEEE-754, y es utilizado internamente por el procesador al realizar cálculos. El procesador convirte cualquier valor, ya sea en precisión simple o doble, a este formato al realizar operaciones, y vuelve a convertir a la precisión de salida deseada al almacenar el resultado en una ubicación de memoria.

Ocupando un total de 80 bits, está compuesto por los siguientes campos:

- Una mantisa de 64 bits, con la particularidad de tener todos sus bits explícitos (a diferencia del formato de precisión simple o doble, donde se asume un primer bit con valor 1).
- Un exponente de 15 bits, con desplazamiento 16383.
- Un indicador de signo de 1 bit.

Un estándar en entornos Unix es hacer el pasaje de valores long double por stack ocupando tres palabras consecutivas de 32 bits (dwords), ocupando un total de 96 bits. Los 16 bits superiores de la dword con la dirección en

memoria más alta se dejan en cero. Como resultado, en estas implementaciones el operador **sizeof(long double)** de C/C++ devuelve 12 en lugar de 10.

Signo	Exponente	Mantisa
$\pm$	$e_{14}e_{13}...e_1e_0$	$m_{63}m_{62}...m_2m_1m_0$

Representación de un número flotante. Un bit para el signo, 15 bits para el exponente y 64 bits para la mantisa.

Teniendo $E = e_{14}e_{13}...e_1e_0$, con este formato pueden representarse los valores a continuación:

E	Mantisa	Valor representado
0	0	± 0
0	$\neq 0$	número denormalizado
$0 < E < 32767$	cualquiera	número en punto flotante normalizado
32767	0	$\pm$ infinito
32767	$\neq 0$	Not A Number

Un número en punto flotante normalizado tiene el bit más significativo de la mantisa m_{63} con el valor 1, y representa al valor real $\pm 1.m_{62}m_{61}...m_2m_1m_0 \times 2^{E-16383}$.

4. Triste intento de cálculo de error relativo de G(x)

$$\begin{aligned}\epsilon(\frac{e^{Q(x)^2}-1}{Q(x)^2}) &= \epsilon(\frac{a}{b}) \\ &\leq \Big|\epsilon(a)\Big| + \Big|\epsilon(b)\Big| + \Big|\epsilon_{(\,\%)}\Big|\end{aligned}$$

$$\begin{aligned}\epsilon(a) &= \epsilon(e^{Q(x)^2}-1) \\ &= \epsilon(a'-b') \\ &\leq \Big|\frac{1}{a'-b'}a'\epsilon(a')\Big| + \Big|\frac{b'}{a'-b'}\epsilon(b')\Big| + \epsilon_{(-)} \\ &= \Big|\frac{1}{a'-1}a'\epsilon(a')\Big| + \Big|\frac{1}{a'-1}\epsilon(1)\Big| + \epsilon_{(-)} \\ &= \Big|\frac{1}{a'-1}a'\epsilon(a')\Big| + \epsilon_{(-)}\end{aligned}$$

$$\begin{aligned}\epsilon(a') &= \epsilon(e^{Q(x)^2}) \\ &= \epsilon(e^{a''}) \\ &\leq \Big|\frac{e^{a''}}{e^{a''}}\epsilon(a'')\Big| + \epsilon_{(exp)} \\ &= \epsilon(a'') + \epsilon_{(exp)}\end{aligned}$$

$$\begin{aligned}
\epsilon(a'') &= \epsilon(Q(x)^2) \\
&= \epsilon(a'''.a''') \\
&\leq \left|\frac{a''' + a'''}{a'''^2} a''' \epsilon(a''')\right| + \epsilon_{(x)} \\
&= \left|2\epsilon(a''')\right| + \epsilon_{(x)}
\end{aligned}$$

$$\begin{aligned}
\epsilon(b) &= \epsilon(Q(x)^2) \\
&= \epsilon(a''') \\
&= \left|2\epsilon(a''')\right| + \epsilon_{(x)}
\end{aligned}$$

$$\begin{aligned}
\epsilon(a''') &= \epsilon(Q(x)^2) \\
&= \epsilon(q_1(x) - q_2(x)) \\
&\leq \left|\frac{1}{q_1(x) - q_2(x)} q_1(x)\epsilon(q_1(x))\right| + \left|\frac{-1}{q_1(x) - q_2(x)} q_2(x)\epsilon(q_2(x))\right| + \epsilon_{(-)}
\end{aligned}$$

Que se me indefine, porque tengo en el denominador a $Q(x)$, que es matemáticamente igual a cero para cualquier valor de x. Esto no sé muy bien como salvarlo.

5. Resultados

5.1. Cálculo de G(x) en distintas precisiones, en un rango amplio de valores de $\mathbb{R}$

Se graficó el exponente del resultado en punto flotante del cálculo de $G(x)$ en la máquina. Esto permite observar y comparar la magnitud de resultados muy grandes y muy pequeños dentro del mismo gráfico, sacrificando poder distinguir el signo del resultado.

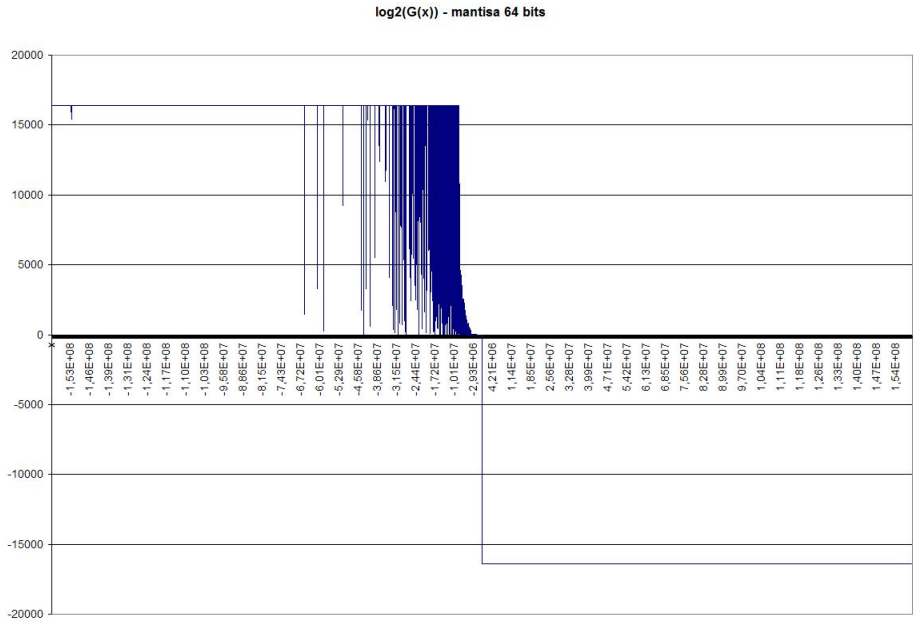

Gráfico 1: Exponente para G(x), con mantisa de 64 bits

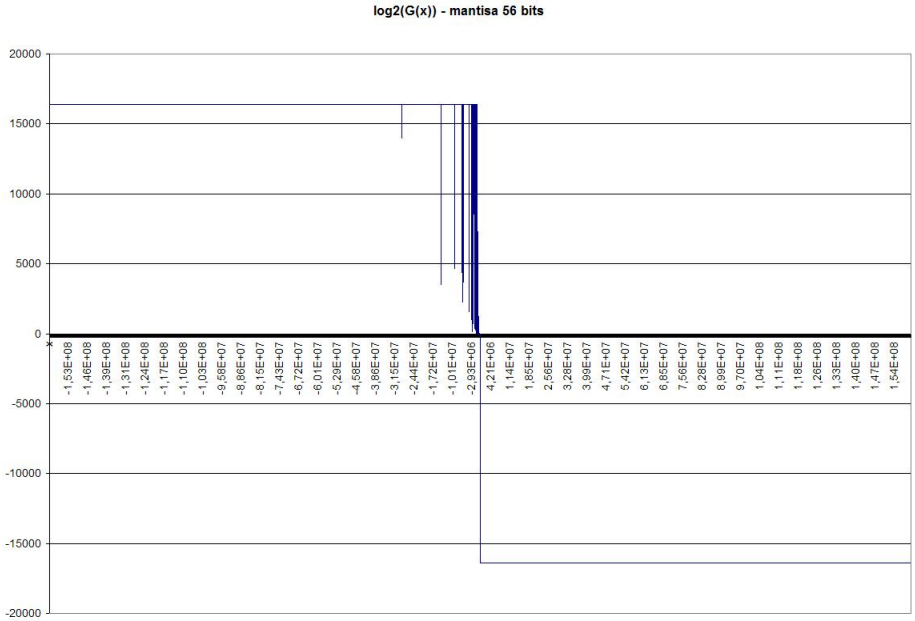

Gráfico 2: Exponente para G(x), con mantisa de 56 bits

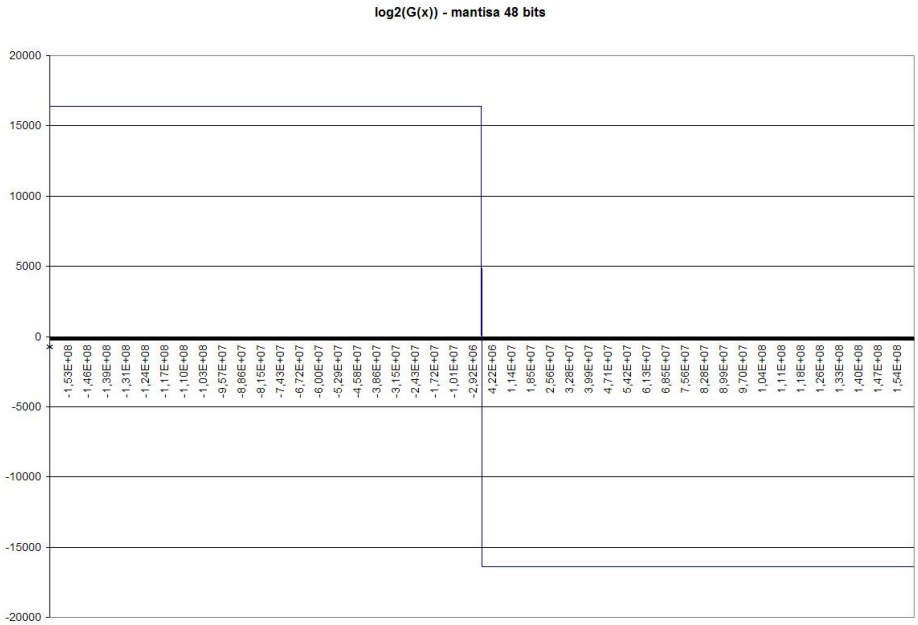

Gráfico 3: Exponente para G(x), con mantisa de 48 bits

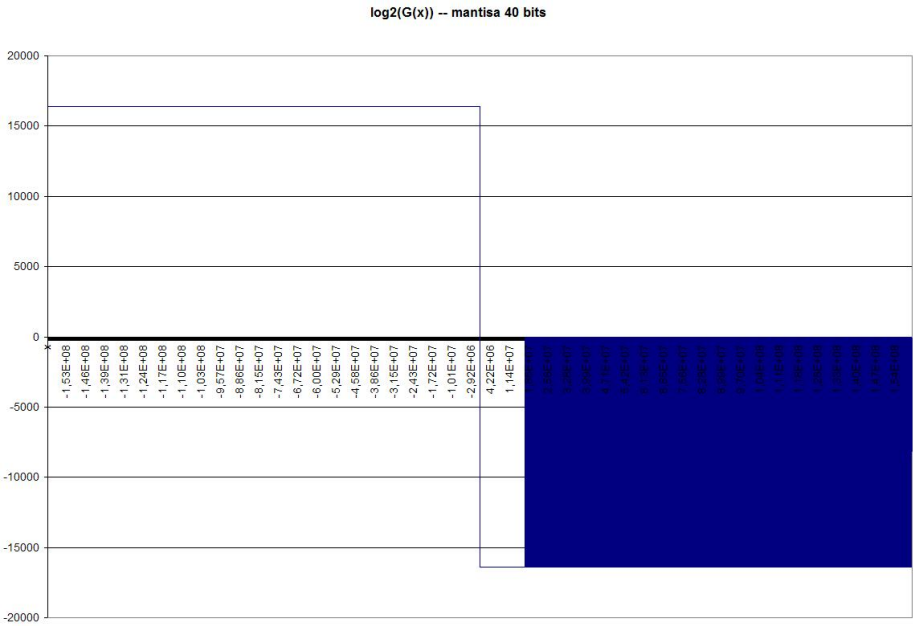

Gráfico 4: Exponente para G(x), con mantisa de 40 bits

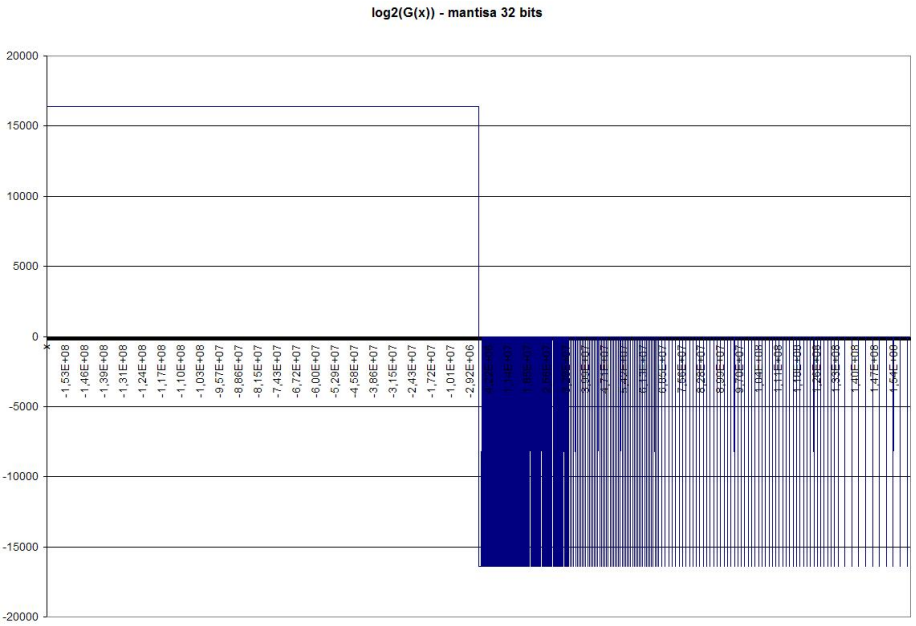

Gráfico 5: Exponente para G(x), con mantisa de 32 bits

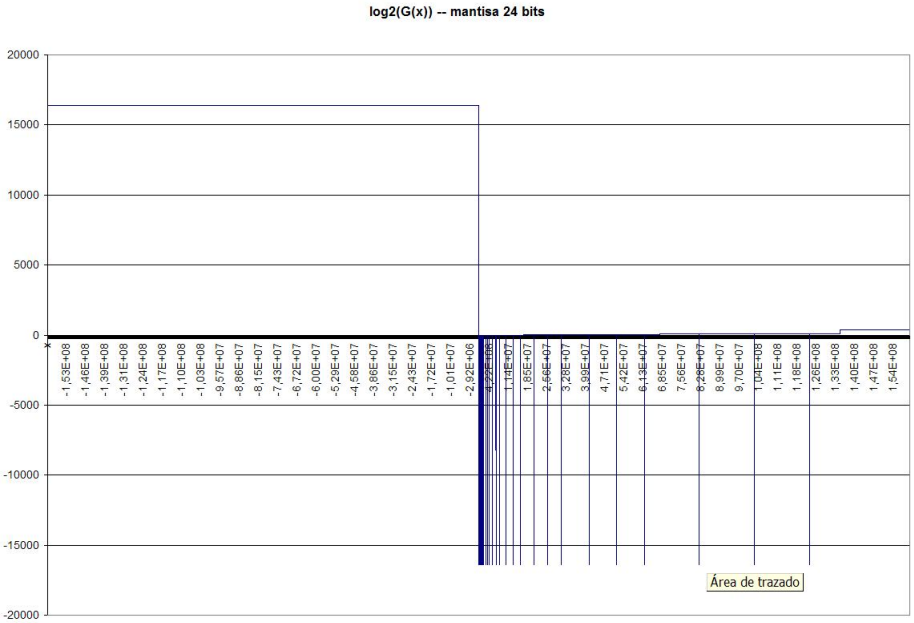

Gráfico 6: Exponente para G(x), con mantisa de 24 bits

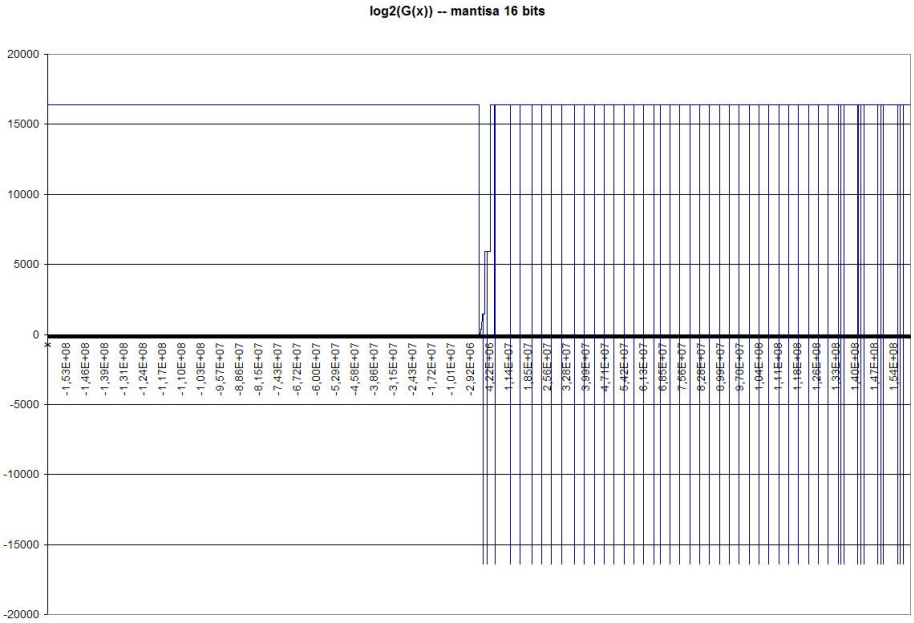

Gráfico 7: Exponente para G(x), con mantisa de 16 bits

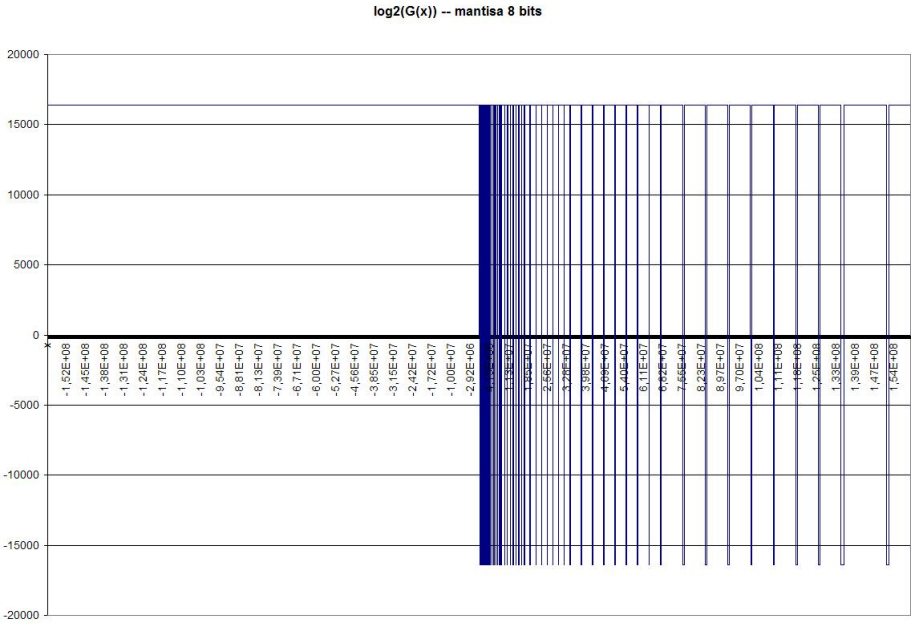

Gráfico 8: Exponente para G(x), con mantisa de 8 bits

5.2.

6. Discusión

1. 1
2. 2

7. Conclusiones

8. Apéndice A

8.1. Enunciado

Laboratorio de Métodos Numéricos - Primer cuatrimestre 2007
Trabajo Práctico Número 1: Sin margen de error (numérico)

El objetivo del trabajo práctico es analizar el comportamiento de la aritmética de punto flotante para el cálculo de una función sobre el conjunto de los números reales. Consideremos la función $G : \mathbb{R} \to \mathbb{R}$ definida por:

$$\begin{aligned} T(x) &= \begin{cases} 1 & \text{si } x = 0 \\ \frac{e^x - 1}{x} & \text{en caso contrario} \end{cases} \\ Q(x) &= \left| x - \sqrt{x^2 + 1} \right| - \frac{1}{x + \sqrt{x^2 + 1}} \\ G(x) &= T(Q(x)^2) \end{aligned}$$

En este trabajo práctico se pide realizar un análisis empírico del comportamiento de la función $G(x)$ para distintos valores de x, calculada con aritmética de punto flotante. Para esto, se pide implementar el cálculo de esta sucesión en aritmética de t dígitos binarios de precisión y, sobre la base de la implementación, realizar las siguientes mediciones:

1. Graficar el valor de $G(x)$ en función de x para diferentes valores de t. ¿Se observa alguna regularidad?

2. Medir el valor de $G(x)$ en función de la cantidad t de dígitos binarios de precisión en la aritmética de punto flotante, para diferentes valores de x. ¿Se observa alguna influencia de la cantidad de dígitos de precisión en el cálculo de $G(x)$?

¿Es posible obtener el valor de $G(x)$ analíticamente? En caso afirmativo, ¿cómo se compara este valor exacto con la aproximación lograda con aritmética de punto flotante? En caso de que se observen diferencias importantes, analizar las causas que generan este comportamiento.

El informe debe contener una descripción detallada de las distintas alternativas que el grupo haya considerado para la implementación de la aritmética de punto flotante de t dígitos binarios de precisión, junto con una discusión de estas alternativas que justifique la opción implementada. Por

otra parte, se debe incluir en la sección correspondiente el código que implementa esta aritmética, junto con todos los comentarios y decisiones relevantes acerca de esta implementación.

Fecha de entrega: Lunes 9 de Abril

9. Apéndice B

9.1. Funciones Relevantes

$$Q(x)$$

```
  pflotante calcularQ(pflotante& paramX) {
pflotante x, raiz, modulo, division;
// calcular fl(x)
x = paramX;
truncar(x);
// calcular x*x
raiz = x*x;
truncar(raiz);
// calcular x*x+1
raiz = raiz + 1;
truncar(raiz);
// calcular sqrt(x*x+1)
raiz = sqrt(raiz);
truncar(raiz);
// calcular x - sqrt(x*x+1)
modulo = x - raiz;
truncar(modulo);
// calcular abs(x - sqrt(x*x+1))
modulo = abs(modulo);
truncar(modulo);
// calcular x + sqrt(x*x+1)
division = x + raiz;
truncar(division);
// calcular 1 / (x + sqrt(x*x+1))
division = 1 / division;
truncar(division);
// valor final
modulo = modulo - division;
truncar(modulo);
// devolver
return modulo;
}
```

...explicación de la implementación del método...

$$T(x)$$

```
  pflotante calcularT(pflotante& paramX) {
pflotante x, cero, resultado, ealax;
// calcular fl(x)
x = paramX;
truncar(x);
// ver si x == 0
cero = 0.0;
truncar(cero);
if (x == cero) {
resultado = 1.0;
}
else {
// calcular e^x
ealax = exp(x);
truncar(ealax);
// calcular e^x - 1
ealax = ealax - 1;
truncar (ealax);
// calcular (e^x - 1)/x
resultado = ealax/x;
}
truncar(resultado);
return resultado;
}
```

...explicación de la implementación del método...

$$G(x)$$

```
  pflotante calcularG(pflotante& paramX) {
pflotante q2, resultado;
// calcular Q(x)^2;
q2 = calcularQ(paramX);
q2 = q2*q2;
truncar(q2);
// calcular T(Q(x)^2)
resultado = calcularT(q2);
truncar(resultado);
return resultado;
}
```

...explicación de la implementación del método...

10. Alguna explicación pertinente

11. Referencias

-
-
-
-