Laboratorio 1: Sistema numérico de punto flotante

Sebastián Valencia Calderón
201111578

1 Introducción

La aritmética de punto flotante, proporciona las bases para la representación de cantidades reales en un computador digital. Dado que los cálculos numéricos son de fundamental importancia para el diseño de sistemas de ingeniería, y solución de grandes y complejos sistemas matemáticos, conviene entender los modos de representación de una cantidad real en un computador, y las implicaciones que esto tiene para los cálculos en cuestión. Las implicaciones, radican en la limitación básica del sistema; un computador digital, permite únicamente la representación de cierto conjunto de números con precisión finita. Esta limitación física para la representación de las cantidades, puede traer errores en los cálculos numéricos ejecutados por algoritmos implementados sobre un lenguaje de programación.

En este caso en particular, se estudia tal representación y sus limitaciones haciendo uso de MATLAB, una poderosa herramienta de computación científica, en la cual resulta pertinente la implementación de algoritmos numéricos, por su facilidad para representar y manipular arreglos de números. Mediante distintas aproximaciones, se pretende conocer las limitaciones de representación de la maquina o mejor, la arquitectura de la maquina sobre los números reales. A través de las implementaciones realizadas, se pretende cumplir con los siguientes objetivos:

- Entender las características, ventajas y desventajas asociadas con las unidades de punto flotante que permiten el desarrollo de la computación numérica.
- Observar casos que ilustran las precauciones que deben ser tenidas en cuenta en sistemas intensivos en computación numérica.

2 Procedimiento

Para cumplir los objetivos enumerados anteriormente, se desarrollan algoritmos para encontrar el epsilon (ϵ_M) de la maquina de un número particular. Con base en esto, se comparan los algoritmos, y de manera análoga, los resultados para un rango grande de los números representados en la máquina donde los algoritmos fueron ejecutados. Después, se procede con la experimentación sobre el rango de números representables, y las cantidades matemáticas fundamentales en MATLAB, posteriormente, se estudian y desarrollan algoritmos numéricos que manipulan la representación de los números para obtener las representaciones numéricas de los resultados ya conocidos de manera analítica.

De manera más desglosada, la ejecución de este laboratorio, procede de la siguiente forma:

1. Implementar un algoritmo numérico que halle de manera eficiente y certera el epsilon de la maquina (ϵ_M), esto es el valor que caracteriza el nivel de precisión de una máquina. El epsilon de la máquina, es el número de punto flotante más pequeño tal que: [1]

$$1 + \epsilon_M > 1$$

   La definición dada anteriormente, es caracterizada únicamente por un número representado en punto flotante. Luego de esto, se pretende generalizar el epsilon sobre cualquier número, es decir, para un número x, $\epsilon(x) = \epsilon \ ; \ |x + \epsilon| > x$. Con una distribución representativa de los números en punto flotante representables por la máquina de trabajo, se pretende comparar distintos algoritmos para su cálculo, y su valor en términos de x.

2. Con base en el estándar IEEE 754-2008 [2], se estudia la representación y de constantes matemáticas de importancia analítica, es decir, la representación numérica de algunos resultados analíticos fundamentales. De la misma manera, se calcula el valor de funciones matemáticas de interés y conocimiento general, esto, con el fin de dilucidar las implicaciones que pueda tener el desarrollo de modelos con base en cálculos numéricos sobre un computador digital.

3 Resultados

A continuación, se evidencian los resultados y la metodología y herramientas de ejecución para cada uno de los problemas propuestos en la guía de laboratorio.

1. "El valor epsilon de una maquina es la diferencia entre 1.0 y el numero de punto flotante siguiente. Ya que este espaciamiento no es uniforme en todo el rango de representación de números de punto flotante, esta definición puede ser extendida alrededor de cualquier otro numero (diferente de 1.0) y escrita como $\epsilon_M(x)$ para cualquier numero de punto flotante x. Esta es una de las razones para calcular el valor de epsilon en MATLAB (el cual cuenta por defecto con un valor constante "eps"), adicionalmente, es conveniente conocer algunos algoritmos para computar este valor, ya que algunos lenguajes de programación no cuentan un valor predefinido del mismo." (Tomado de la guía de laboratorio).

   En el script 4, se evidencia el diseño y escritura de un algoritmo para calcular ϵ_M. El script, muestra el algoritmo propuesto en el enunciado. Después de su ejecución, el valor fue:

$$\epsilon_M = 2.220446049250313 \times 10^{-16}$$

   La constante eps de MATLAB, es: $2.220446049250313 \times 10^{-16}$.

2. Para generalizar la definición de ϵ_M, sobre cualquier número, se introduce el concepto de $\epsilon_M(x)$, es decir, el número en punto flotante más pequeño para el cual resulta que sumarlo a x, se tiene un número mayo a x en la representación del computador. Es decir, $\epsilon(x) = \epsilon \ ; \ |x + \epsilon| > x$. Para diseñar un algoritmo que compute esto, resulta

pertinente recurrir al script 4, en este, se cuenta con un valor de inicializacion de ϵ, de manera que $1.0 + epsilon > 1.0$. Para la generalización requerida, se debe tener un ϵ, tal que $x + \epsilon > x$, de manera que se doble el valor de ϵ, y se incremente la suma $x + \epsilon$ hasta que $x + \epsilon \leq x$, se ha encontrado el valor $2 \times \epsilon$. En el script 5, se evidencia el desarrollo de un algortimo, que análogamente al enunciado anteriormente, reduce el valor de ϵ, hasta que $x + \epsilon \leq x$.

Script 1: Gráfica de x vs $\epsilon(x)$. (plotEpsilon.m)

```
% Plots the value of the machine epsilon around a number

x = 0:100000;
x = x.^100;

y = MaqEps(x);

plot(x, y);
set(gcf, 'Position', [400 400 700 700]);
saveas(gcf, '../img/plot_epsilon.png');

eps_max = max(y);
eps_min = min(y);

range = eps_max - eps_min;
```

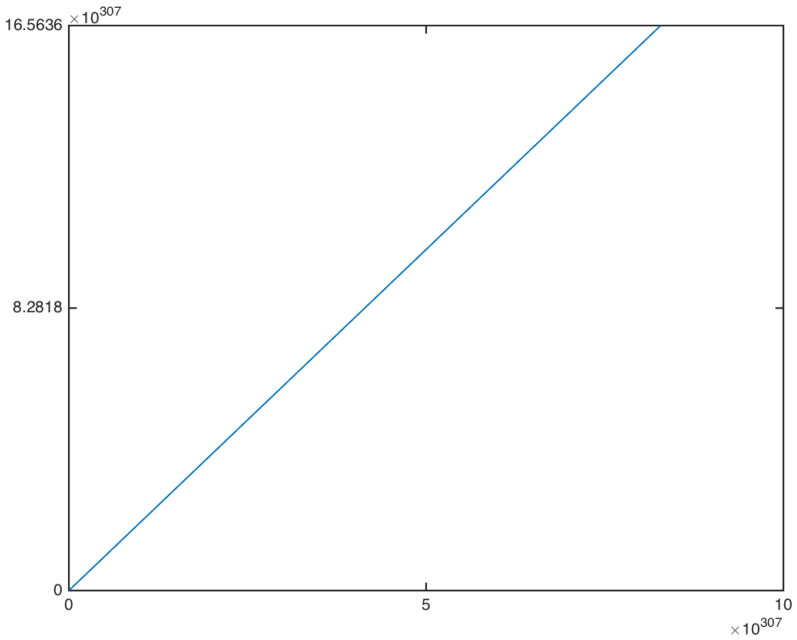

Figura 1: Gráfica de x vs $\epsilon_M(x)$

En la gráfica anterior, puede verse que entre más grande sea x, más grande es su

epsilon definido como se definió anteriormente. El crecimiento de este patrón es lineal, lo cual es determinado por la representación de los números reales en un computador.

3. El procedimiento diseñado por William Kahan, y utilizado en el paquete LINPACK, implementado en FORTRAN 77 [3], tiene como objetivo estimar las unidades de redondeo en una cantidad proporcional al tamaño de x. Es decir, el epsilon para un x en punto flotante. Sin embargo, asume ciertas características, que son riesgosas si no se usa este de manera adecuada. Por ejemplo, la base usada para la representación en punto flotante no debe ser potencia de tres. El código original, puede verse en el script 6.

   Para analizar el desempeño de este y comparar sus resultados con los implementados previamente, se procede a implementar el algoritmo en MATLAB, realizar pequeños experimentos para comparar los valores, y posteriormente, realizar la prueba sobre un rango representativo de los números de punto flotante capaces de ser representados por la máquina digital que corre los programas.

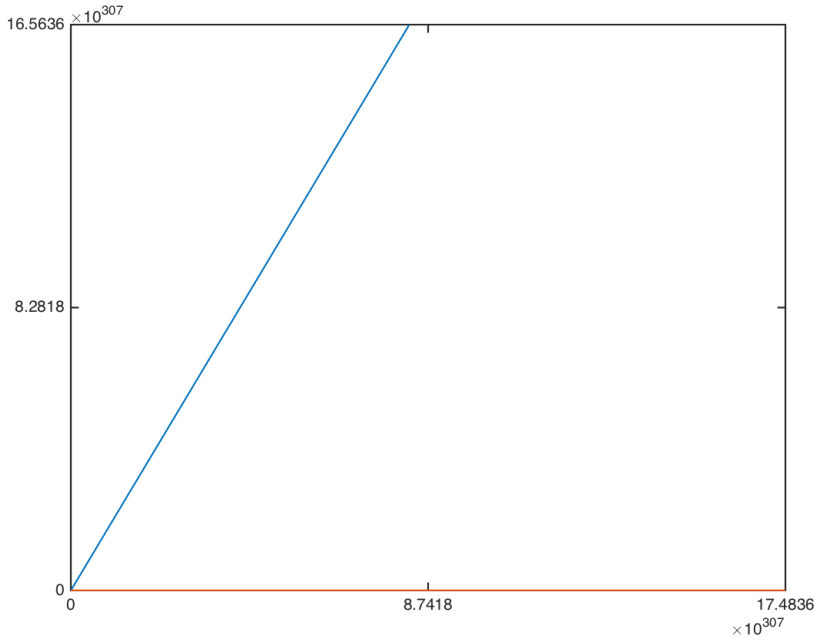

Figura 2: Gráfica de x vs $\epsilon_M(x)$, los distintos trazos, muestran el epsilon de Kahan y el implementado anteriormente.

Puede verse en la gráfica anterior, que la precisión del algoritmo de Kahan, favorece la implementacion y análisis de resultados de los algoritmos numéricos desarrollados. Sin embargo, si estos resultados se cotejan con la arquitectura de la maquina, estos carecen de sentido. La diferencia en magnitud de ambos valores es demasiado grande, y se sabe que por la representación de los números dentro de un computador digital, a medida que va creciendo un numero, la diferencia entre el siguiente numero mas grande representado y el mismo va aumentando. Se determina que es mas confiable el algoritmo implementado. Además, no tiene restricciones.

4. Después de investigar el estándar de la IEEE [2], para la representación y manipulación de los números reales en un computador digital, puede decirse de las siguientes constantes lo siguiente:

   - *eps*. Su valor es 2.220446049250313e-16. Este valor, es la mínima distancia reconocida entre dos números de punto flotante x y y. Es decir, la diferencia entre un numero x en punto flotante según la representación definida por el estándar de la IEEE, y el siguiente numero que la maquina puede representar bajo el mismo estándar.
   - $realmax$. Su valor es 1.797693134862316e+308. Es el valor máximo representable en punto flotante según el estándar de la IEEE para la representación de los mismos.
   - $realmin$. Su valor es 2.225073858507201e-308. Es el valor mínimo representable en punto flotante según el estándar de la IEEE para la representación de los mismos. Claro está, sobre la máquina que corre el algoritmo.
   - $1/realmin$. Su valor es 4.494232837155790e+307. Representa valores anormales según la definición del estándar de la IEEE.
   - $1/realmax$. Su valor es 5.562684646268003e-309. Representa valores anormales según la definición del estándar de la IEEE.

5. A continuación, se brinda el valor de MATLAB para los valores indeterminados, y se brinda una posible explicación.

   - $\frac{0}{0}$
   - $\frac{\infty}{\infty}$
   - $\infty * 0$
   - 1^{∞}
   - $\infty - \infty$
   - 0^{∞}
   - ∞^{0}

   Las operaciones listadas anteriormente, están definidas en el estándar de la IEEE [2]. Los últimos dos, poseen un valor definid por el estándar siguiendo los preceptos básicos del análisis real: "todo número elevado a la cero es uno", "cero elevado a cualquier cosa es cero". Sin embargo, los demás se encuentran definidos en el estándar haciendo uso de la constante NaN.

   El símbolo NaN, denota Not a Number. Este símbolo, representa los valores que analíticamente o numéricmente no representa un número real. Su representación es a través de la cadena de bits con todos los exponentes en uno, y ceros en la mantisa. Los valores $\frac{0}{0}$, $\frac{\infty}{\infty}$, $\infty * 0$, 1^{∞}, $\infty - \infty$, denotan en MATLAB, y en el estándar el valor NaN. [2], [4].

6. Dado que analíticamente $sin(\pi) = 0 \wedge cos(\pi) = -1 \wedge sin(\pi)^2 + cos(\pi)^2 = 1$, se investiga su valor en MATLAB, y se explica este valor.

- $sin(\pi)$. En MATLAB, su valor es 1.224646799147353e-16, esto se debe a la representción de π en el computador, es decir, la forma de calcularlo (es necesario recordar que se trata de un valor irracional, y sus dígitos son infinitos.), y a la representación o algoritmo utilizado para encontrar sin, de un valor.
- $cos(\pi)$. En MATLAB, su valor es -1, lo cua puede deberse a un cortocircuito en la implementación interna del algoritmo para encontrar cos de un valor dado.
- $sin(\pi)^2 + cos(\pi)^2$. En MATLAB, su valor es 1. Lo cual se debe a que se está sumando un valor cercano a cero con uno, y la diferencia entre estos dos valores es menor al epsilon de la máquina.

7. A continuación, se listan y explican los cálculos propuestos en el contexto del estándar IEEE 754-2008, [2].

   - $(1 + 1E - 16) - 1 \sim 0$
   - $(1 + 2E - 16) - 1 \sim 2.2204E - 16$
   - $(1 - 1E - 16) - 1 \sim -1.1102E - 16$
   - $1 + (1E - 16 - 1) \sim 1.1102E - 16$

8. Para encontrar el valor numérico de la expresión analítica definida como $\sum_{k=1}^{3000} \frac{1}{k^2} = 1.6446$, en MATLAB con cuatro cifras significativas, se presenta una alternativa. La alternativa propuesta en la guía, corta en cada paso el valor acumulado de la suma. El script propuesto, se muestra a continuación.

   `Script 2: Cálculo de` $\sum_{k=1}^{3000} \frac{1}{k^2}$`, usando redondeo en cada etapa intermedia. El resultado de ésta computación, varía mucho sobre el deseado. (badSummation.m)`

```
sum = 0;

for k=1:3000
    sum = chop(sum+1/k^2, 4);
end

sum
```

   El valor resultante de este cálculo, es 1.6240, el cual está muy lejos del valor analítico deseado. Para esto, se propone la siguiente solución:

   `Script 3: Cálculo de` $\sum_{k=1}^{3000} \frac{1}{k^2}$`, usando redondeo en el resultado final. El resultado de ésta computación, es parecido al valor analítico del mismo. (summation.m)`

```
sum = 0;
for k=1:3000
    k = k * k;
    sum = sum + (1 / k);
end

```

```
7 chop(sum, 5)
```

La cancelación en el script 2, se da por que es en cada etapa intermedia, interfiriendo los resultados del cálculo en cada iteración, es decir, el valor acumulado varía mucho en cada redondeo, pues se suma el valor actual con la nueva expresión para calcular el nuevo valor, pero se redondea la suma. De tal manera, el resultado nuevo ya está redondeado. En el procedimiento propuesto (script 3), se redondea el valor hasta lo ultimo, cuando ya todo el cálculo numérico, haya sido consumido por la iteración. El valor resultado del último método listado es: 1.6446.

9. La constante e, definida como 2.7182818, es definida analíticamente como un límite al infinito de una expresión aritmética. Ésta expresión, sugiere una aproximación para su implementación por computador. Sin embargo, el planteamiento debe lo suficientemente bueno como para brindar eficiencia y precisión. A continuación, se muestran la definición analítica, y la aproximación por computador.

$$e = \lim_{n\to\infty} \left(1 + \frac{1}{n}\right)^n \Rightarrow e \sim \left(1 + \frac{1}{n}\right)^n \quad n \text{ suficientemente grande}$$

Dada la anterior definición, se procede a diseñar un Script en MATLAB para encontrar un valor numérico a tal constante. El script 9, muestra el algoritmo diseñado para encontrar de manera numérica e iterativa un valor aproximado de e. Este da pasos de 100, en 100 hasta que la diferencia entre el valor hallado y el real convenga con un nivel de tolerancia definido por la entrada de la función. Experimentado con esta función, se encuentra que para niveles de tolerancia significativamente grandes, se tiene que el valor hallado es muy lejano al real, sin embargo, con niveles de tolerancia entre 0.1 y 0.000001, el valor encontrado por el algoritmo es cercano al real. Mientras más el nivel de tolerancia se acerca a 0.000001, se tiene un valor más cercano al real. Al pasar este limite (0.000001), la precision requerida por el computador, no alcanza a cubrir los requerimientos de la maquina. En este ultimo caso, el valor resultado del cálculo es NaN. Sobre la eficiencia, puede verse que el algoritmo n itera sobre todos los números hasta alcanzar convergencia, sino da pasos de 100 en 100 hasta lograr esto.

4 Conclusiones

Por medio del desarrollo de los ejercicios numéricos propuestos, se determinó la importancia de contemplar la presencia del error en los cálculos que apoyan los modelos, diseños y simulaciones realizadas en ingeniería. Ya que el análisis numérico y la computación científica desempañan un papel muy importante en el desarrollo moderno de la ciencia y la ingeniería, es importante saber que las máquinas digitales que corren los algoritmos numéricos son maquinas de recursos limitados para la representación y manipulación de números reales y enteros; los cual tiene una importancia demasiado grande a la hora de solucionar problemas usando un computador. A manera de lista, se enuncian los principales hallazgos del desarrollo de la guía.

- La presencia de errores numéricos, puede llevar a grandes problemas si estos no se contemplan en el diseño de una solución por computador. Es necesario tener cuidado al comparar expresiones analíticas con expresiones numéricas.

- La solución de sistemas numéricos por medio de computador, depende demasiado de la arquitectura y especificaciones técnicas de la máquina que corre los algoritmos. De la misma manera, el lenguaje de programación, la eficiencia y presicion de los algoritmos, son aspectos de fundamental importancia al evaluar una solución numérica y por computador de un problema.

- El estándar IEEE amplia-mente citado en el desarrollo de la, sienta una base solida para el desarrollo de la arquitectura de un computador y el tratamiento de los tipos de datos numéricos en un lenguaje de programación. Este busca ordenar y definir la representación de los números según la arquitectura de un computador y constituye una guía para los implementadores de las arquitecturas y los lenguajes de programación, las principales herramientas de trabajo de un analista. Es de fundamental importancia la existencia de estos estándares para minimizar la importancia de la maquina que corre los algoritmos.

5 Bibliografía

[1] Forsythe, G.E. and Malcolm, M.A. and Moler, C.B. *Computer methods for mathematical computations*, Prentice-Hall series in automatic computation. 1977.

[2] IEEE Task P754. *IEEE 754-2008, Standard for Floating-Point Arithmetic*, pub-IEEE-STD. 2008.

[3] Dongarra, J.J. and Bunch, J.R. and Moler, C.B. and Stewart, G.W. *LINPACK Users' Guide*, Society for Industrial and Applied Mathematics. 1979.

[4] Stallins, W. *Computer Organization and Architecture*, Pearson Education. 2013.

6 Scripts

Script 4: Cálculo de ϵ_M. (MaqEpsOne.m)

```
%{
    This script, computes dynamically the value of the machine
    epsilon, executed in MATLAB/Octave. For this purpose, we use
    the fact that the floating point representation of 1.0 + eps,
    must be bigger tan one. Then we decrease the value of eps by
    a factor of (1 / 2.0), to find the required eps.

    eps = epsilon
%}

format long;

epsilon = 1.0;
fl = 1.0 + epsilon;

while fl > 1.0
    epsilon = epsilon / 2.0;
    fl = 1.0 + epsilon;
end;

% We find the proper value, since the condition
% at the while loop increase the value of epsilon

epsilon = abs(2.0 * epsilon);
disp(epsilon);
```

Script 5: Cálculo de ϵ_M para un x específico. (MaqEps.m)

```
function [eps] = MaqEps(x)

    eps = x;
    fl = x + eps;

    while fl > x
        eps = eps / 2.0;
        fl = x + eps;
    end;

    eps = abs(2.0 * eps);
end
```

Script 6: Cálculo de $\epsilon_M(x)$ según el algoritmo de Kahan implementado en FORTRAN. (kahan.f)

```
REAL FUNCTION EPSLON(X)
      REAL X
      REAL a , b , c , eps

      a = 4.0/3.0
      DO WHILE ( .TRUE. )
         b = a - 1.0
         c = b + b + b
         eps = ABS(c-1.0)
         IF ( eps.NE.0.0 ) GOTO 1
      ENDDO
 1    EPSLON = eps*ABS(X)
      CONTINUE
      END
```

Script 7: Cálculo de ϵ_M para un x específico. (KahanEpsilon.m)

```
function [epsilon] = KahanEpsilon(x)
    a = 4.0/3.0;
    eps = 0.0;
    while(true)
        b = a - 1.0;
        c = b + b + b;
        eps = abs(c - 1.0);
        if(eps ~= 0.0)
            break;
        end;
    end;

    epsilon = eps * abs(x);
end
```

Script 8: Cálculo de ϵ_M para un x específico. (plotEpsilonComp.m)

```
% Plots the value of the machine epsilon around a number

x = 0:100000;
x = x.^100;

y = MaqEps(x);
z = KahanEpsilon(x);

plot(x, y, x, z);
```

```
set(gcf, 'Position', [400 400 700 700]);
saveas(gcf, '../img/plot_epsiloncomp.png');
```

Script 9: Función en MATLAB para encontrar de manera numérica el valor de e con cierta tolerancia o diferencia absoluta con el valor real (el definido en MATLAB). (FindExp.m)

```
function [estimate, k] = FindExp(tolerance)
    k = 10;
    real = exp(1);
    estimate = (1 + (1 / k)) ^ k;

    while(abs(estimate - real) > tolerance)
        k = k * 100;
        estimate = (1 + (1 / k)) ^ k;
    end

end
```

Lista de Scripts