Tema 1 - Representación de números de punto flotante

Curso de Física Computacional

M. en C. Gustavo Contreras Mayén

Contenido

Contenido

Contenido

Contenido

Contenido

¿Qué es lo que pasa?

Realiza la siguiente operación en Python:

```
>>> 1 - 0.9 - 0.1
```

¿Qué es lo que pasa?

Realiza la siguiente operación en Python:

```
>>> 1 - 0.9 - 0.1
-2.7755575615628914e-17
```

¿Qué es lo que pasa?

Realiza la siguiente operación en Python:

```
>>> 1 - 0.9 - 0.1
-2.7755575615628914e-17
```

Pregunta

¿Por qué el resultado que esperamos no es cero?

Considera el siguiente código

```
1 suma = 1
2 for i in range(0,1000000):
3     suma = suma+0.0000001
4
5 print (suma)
```

¿Cuál es el resultado que nos devuelve el programa?

Considera el siguiente código

```
1 suma = 1
2 for i in range(0,1000000):
3     suma = suma+0.0000001
4
5 print (suma)
```

¿Cuál es el resultado que nos devuelve el programa?

1.1000000000583867

Considera el siguiente código

```
1 suma = 1
2 for i in range(0,1000000):
3     suma = suma+0.0000001
4
5 print (suma)
```

¿Cuál es el resultado que nos devuelve el programa?

1.1000000000583867

Pregunta

¿En dónde está el error?

Para dar respuesta a las preguntas, tendremos que ahondar en la manera en que se representan los números en una computadora.

Contenido

Código binario

La forma de almacenar un número en una computadora es mediante el sistema binario (código binario), es decir, sólo utiliza el 0 y el 1 para representar cualquier número.

Código binario

Según sea el diseño de la arquitectura y el hardware de una computadora, será la precisión con la que se represente un número.

Por ejemplo, para una computadora que utiliza 8 bytes, es decir, 64 bits para representar un entero, el primer bit lo utiliza para el signo y los restantes 63 para el número, así el máximo valor que se puede representar será

$$\sum_{k=0}^{62} 2^k$$

El formato para la representación de un número real en una computadora depende del diseño de hardware y software. El formato común es el de *punto flotante*, donde se le asignan ciertos bits para guardar el exponente y el resto para la mantisa.

La razón más común de los errores en una computadora se atribuyen al error de representar un número real mediante un número limitado de bits, comúnmente se denominan errores de redondeo al guardar un número en la memoria.

El épsilon, ϵ, de la computadora es lo que determina la precisión con la que se representa un número, como ya vimos, se define como el tamaño del intervalo entre 1 y el siguiente número mayor que 1 distinguible de 1.

Esto significa que ningún número entre 1 y $1 + \epsilon$ se puede representar en la computadora.

Por ejemplo, cualquier número $1 + \alpha$ se redondea a 1 si $0 < \alpha < \epsilon/2$, o se redondea a $1 + \epsilon$ si $\epsilon/2 \leq \alpha$.

Por lo que se puede considerar que $\epsilon/2$ es el máximo error posible de redondeo para 1. En otras palabras, cuando se halla 1.0 en la memoria de la computadora, el valor original pudo estar entre $1 - \epsilon/2 < x < 1 + \epsilon/2$.

Finalmente se puede concluir que el error de redondeo implicado al guardar cualquier número real $R \in \mathbb{R}$ en la memoria de una computadora, es aproximadamente igual a $\epsilon R/2$, si el número se redondea por exceso y ϵR si se redondea por defecto.

Operaciones aritméticas en binario

Las cuatro operaciones aritméticas básicas: suma, resta, multiplicación y división en notación binaria presentan dos tipos de situaciones en los que aparecen muchos errores de redondeo:

- Cuando se suma o se resta un número muy pequeño con uno muy grande.
- Cuando se divide entre un número pequeño o se multiplica por un número muy grande.

Contenido

Números en representación de punto flotante

La forma estándar de representar un número real en forma decimal es con una parte entera, un punto decimal y una parte fraccionaria, por ejemplo 45.67432 o 0.00012534.

Números en representación de punto flotante

La forma estándar de representar un número real en forma decimal es con una parte entera, un punto decimal y una parte fraccionaria, por ejemplo 45.67432 o 0.00012534.

Otra forma estándar, llamada *notación científica normalizada*, en la cual la parte entera es cero y se obtiene al multiplicar por una potencia adecuada de 10.

Se tiene que 45.67432 se puede representar en notación científica normalizada como 0.4567432×10^2 y 0.0012534 como 0.12534×10^{-2}.

Representación de punto flotante

En notación científica, un número se representa por una fracción multiplicada por una potencia de 10 y el primer número a la derecha del punto decimal es diferente de cero. En computación a la notación científica normalizada se le llama *representación de punto flotante*.

Un sistema computacional representa a los números en punto flotante con las limitaciones que impone una longitud finita de palabra. Una computadora puede tener una longitud de palabra de 64 bits (dígitos binarios) la cual puede representar números binarios en la forma

$$x = \pm q \times 2^m$$

$$x = \pm q \times 2^m$$

donde

- el signo de x ocupa 1 bit.
- el signo de m ocupa 1 bit.
- el entero $|m|$ ocupa 10 bits.
- el número q ocupa 52 bits.

Al número q se le denomina *mantisa* y al número m se le llama *exponente*. Si se supone que m se puede representar con a lo más 10 bits entonces el máximo valor de m que se puede representar es $2^{10}–1 = 1023$.

Números de máquina

Los números reales representables en una computadora se llaman *números de máquina*. Para poder representar un número

$$x = 0.d_1 d_2 d_3 \ldots \times 10^m$$

como número de máquina, denotado por $fl(x)$, se obtiene cortando la mantisa de x en k cifras.

Existen dos formas de efectuar ese corte:

1. La primera es simplemente eliminando los dígitos a partir de d_{k+1}. Esta forma recibe el nombre de truncamiento.
2. La segunda forma es determinar a d_k a partir de d_{k+1}, si $d_{k+1} < 5$, d_k queda sin cambio. En caso contrario, aumentamos en uno d_k, esto se conoce como redondeo.

Contenido

Propagación de errores

Para investigar la propagación del error en cálculos sucesivos se consideran dos números p y q junto con sus aproximaciones $\widehat{p}$ y $\widehat{q}$ con errores ϵ_p y ϵ_q, respectivamente. Entonces

$$p + q = (\widehat{p} + \epsilon_p) + (\widehat{q} + \epsilon_q) = (\widehat{p} + \widehat{q}) + (\epsilon_p + \epsilon_q)$$

por lo que el error de la suma, es la suma de los errores y el error relativo está dado por:

$$\frac{(p + q) - (\widehat{p} + \widehat{q})}{(p + q)} = \frac{\epsilon_p + \epsilon_q}{p + q}$$

Para la multiplicación se tiene que:

$$pq = (\widehat{p} + \epsilon_p)(\widehat{q} + \epsilon_q) = \widehat{p}\widehat{q} + \widehat{p}\epsilon_q + \widehat{q}\epsilon_p + (\epsilon_p + \epsilon_q)$$

Por lo que si $|p|$, $|q| > 1$, los términos $\widehat{p}\epsilon_q$ y $\widehat{q}\epsilon_p$ muestran un aumento en los errores originales.

Para tener una mejor visión de la multiplicación, se calcula el error relativo, bajo las suposiciones: $p, q \neq 0, \frac{\widehat{p}}{p} \simeq 1, \frac{\widehat{q}}{q} \simeq 1, \frac{\epsilon_p}{p}\frac{\epsilon_q}{q} = E_p E_q \simeq 0$. Se tiene que

$$\frac{pq - \widehat{p}\widehat{q}}{pq} = (\widehat{p} + \epsilon_p + \epsilon_p)(\widehat{q} + \epsilon_q) = \frac{\epsilon_q}{q} + \frac{\epsilon_p}{p} + E_p E_q \simeq$$

$$\simeq \frac{\epsilon_q}{q} + \frac{\epsilon_p}{p} = E_p + Eq$$

El error relativo en el producto es aproximadamente la suma de las razones entre los errores y las aproximaciones.

Casos especiales del IEEE 754

Tanto en los tipos de datos de precisión simple como de precisión doble, existen algunos casos especiales que dependen de los valores del signo, del exponente y de la mantisa.

Signo (s)	**Exponente (exp)**	**Mantisa (m)**	**Significado**
Positivo (0)	Todos unos $(111\ldots11)$	Todos ceros $(000\ldots00)$	Más infinito $(+\infty)$
Negativo (1)	Todos unos $(111\ldots11)$	Todos ceros $(000\ldots00)$	Menos infinito $(-\infty)$
0 ó 1	Todos unos $(111\ldots11)$	Distinta de todo ceros	No es un número (Not a Number, **NaN**)
0 ó 1	Todos ceros $(000\ldots00)$	Todos ceros $(000\ldots00)$	Representa al cero (0)
0 ó 1	Todos ceros $(000\ldots00)$	Distinta de todo ceros	Número muy pequeño cercano al cero

Valores de punto flotante en python.

```
1 import sys
2
3 print(sys.float_info)
```

¿Y si la representación del número binario es periódica?

Hagamos la conversión de 0.4 a binario:

$$
\begin{aligned}
0.4 \times 2 &= 0.8 && \rightarrow a_{.1} = 0 \\
0.8 \times 2 &= 1.60 && \rightarrow a_{.2} = 1 \\
0.6 \times 2 &= 1.20 && \rightarrow a_{.3} = 1 \\
0.2 \times 2 &= 0.4 && \rightarrow a_{.4} = 0 \\
0.4 \times 2 &= 0.8 && \rightarrow a_{.5} = 0 \\
0.8 \times 2 &= 1.60 && \rightarrow a_{.6} = 1 \\
0.6 \times 2 &= 1.20 && \rightarrow a_{.7} = 1 \\
0.2 \times 2 &= 0.4 && \rightarrow a_{.8} = 0 \\
0.4 \times 2 &= 0.8 && \rightarrow a_{.9} = 0
\end{aligned}
$$

$$
\begin{aligned}
0.8 \times 2 &= 1.60 && \rightarrow a_{.10} = 1 \\
0.6 \times 2 &= 1.20 && \rightarrow a_{.11} = 1 \\
0.2 \times 2 &= 0.4 && \rightarrow a_{.12} = 0 \\
0.4 \times 2 &= 0.8 && \rightarrow a_{.13} = 0
\end{aligned}
$$

Como vemos, al expresar el valor de 0.4 a binario, tenemos un número periódico, en donde 0011 se repite indefinidamente. En el ejemplo se llegó hasta 0.01100011001100 que es igual a 0.39990234375.

Cuanto más se repita el cálculo, más nos acercamos a 0.40, pero como tenemos un número binario periódico, no importa la cantidad de dígitos fraccionarios, siempre se acercará a 0.40, por lo que se redondea luego de cierto número de dígitos.

Veamos la conversión a decimal:

$$
\begin{aligned}
0.0110011001100 = {} & (0 \times 2^{0}) + (0 \times 2^{-1}) + (1 \times 2^{-2}) + (1 \times 2^{-3}) + \\
& + (0 \times 2^{-4}) + (0 \times 2^{-5}) + (1 \times 2^{-6}) + (1 \times 2^{-7}) + \\
& + (0 \times 2^{-8}) + (0 \times 2^{-9}) + (1 \times 2^{-10}) + (1 \times 2^{-11}) + \\
& + (0 \times 2^{-12}) + (0 \times 2^{-13}) = \\
& = 0.39990234375
\end{aligned}
$$

Que se puede redondear a 0.4

Ejercicio de tarea

Elabora un programa utilizando la aritmética y código de `python`para convertir números de punto flotante a binario (tanto la parte entera como la parte fraccionaria)

Ejercicio 1

Aprendimos en la secundaria a resolver la ecuación homogénea de segundo grado:

$$ax^2 + bx + c = 0$$

que tiene una solución analítica que se puede escribir como

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \qquad x_{1,2} = \frac{-2c}{b \pm \sqrt{b^2 - 4ac}}$$

Revisando la expresión anterior vemos que la cancelación de la diferencia (y por tanto, un incremento en el error) aumenta cuando $b^2 >> 4ac$ debido a que la raíz cuadrada y el siguiente término están muy próximas a cancelarse.

Manos al código

1. Escribe un programa que calcule las cuatro soluciones para valores arbitrarios de a, b y c.

Manos al código

1. Escribe un programa que calcule las cuatro soluciones para valores arbitrarios de a, b y c.
2. Revisa cómo los errores obtenidos en los cálculos, aumentan conforme hay una cancelación de la diferencia de términos y su relación con la precisión de la máquina. Prueba con los siguientes valores $a = 1$, $b = 1$, $c = 10^{-n}, n = 1, 2, 3, \ldots$

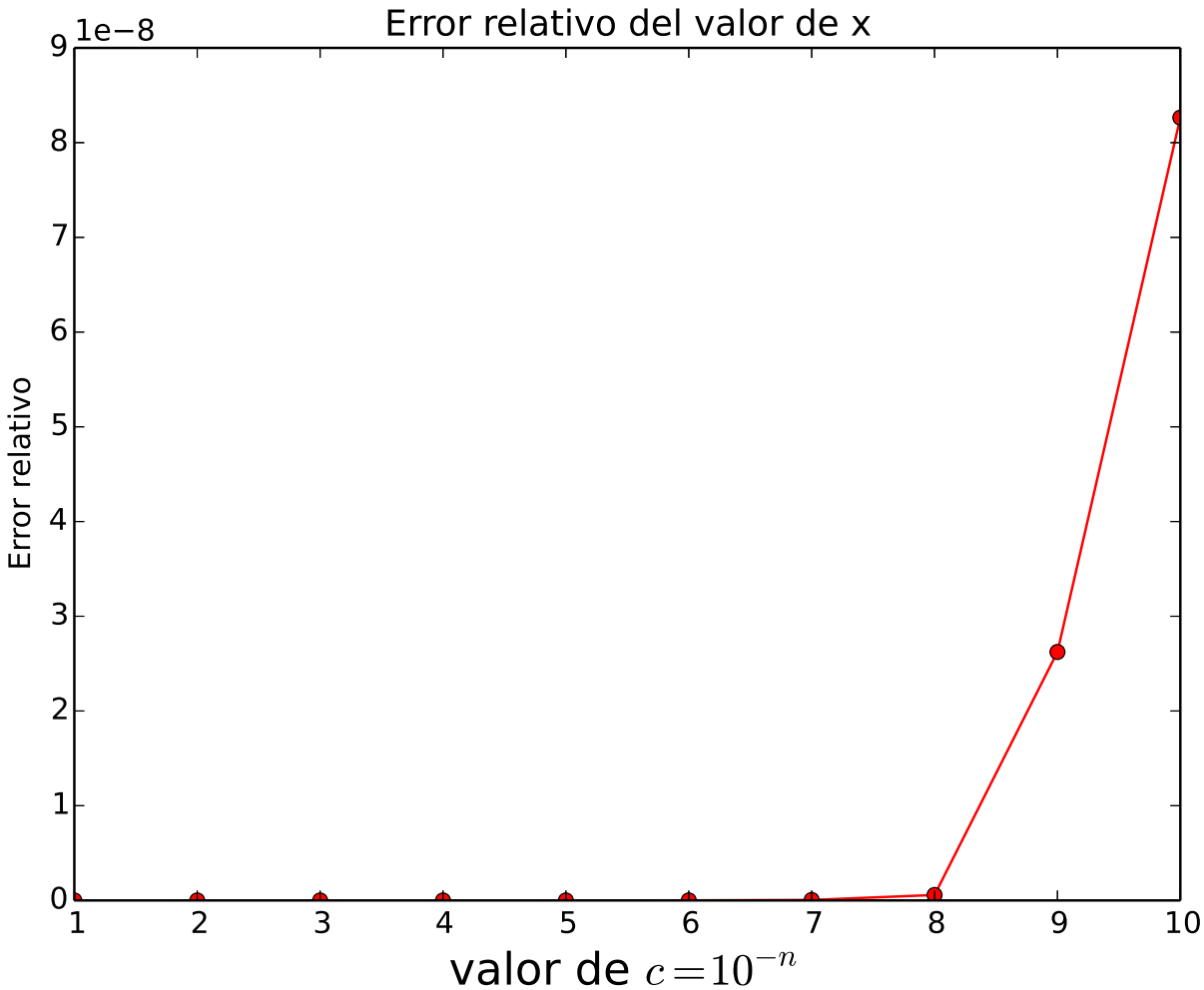
Manos al código

1. Escribe un programa que calcule las cuatro soluciones para valores arbitrarios de a, b y c.
2. Revisa cómo los errores obtenidos en los cálculos, aumentan conforme hay una cancelación de la diferencia de términos y su relación con la precisión de la máquina. Prueba con los siguientes valores $a = 1$, $b = 1$, $c = 10^{-n}, n = 1, 2, 3, \ldots$
3. Cómo mejorarías el programa para obtener la mayor precisión en tu respuesta?

¿Qué resultados debemos esperar?

Al momento de sentarse a escribir el código, hay que contemplar o estimar, qué resultados son los que deberíamos de esperar, con el ejercicio, para que valor de $c = 10^{-n}$, obtenemos un error relativo, y pues al variar el valor de $n = 1, \ldots, 9, 10$ podremos obtener una idea del comportamiento del error, si graficamos los valores del error relativo contra n.

Gráfica del error relativo y el valor de c

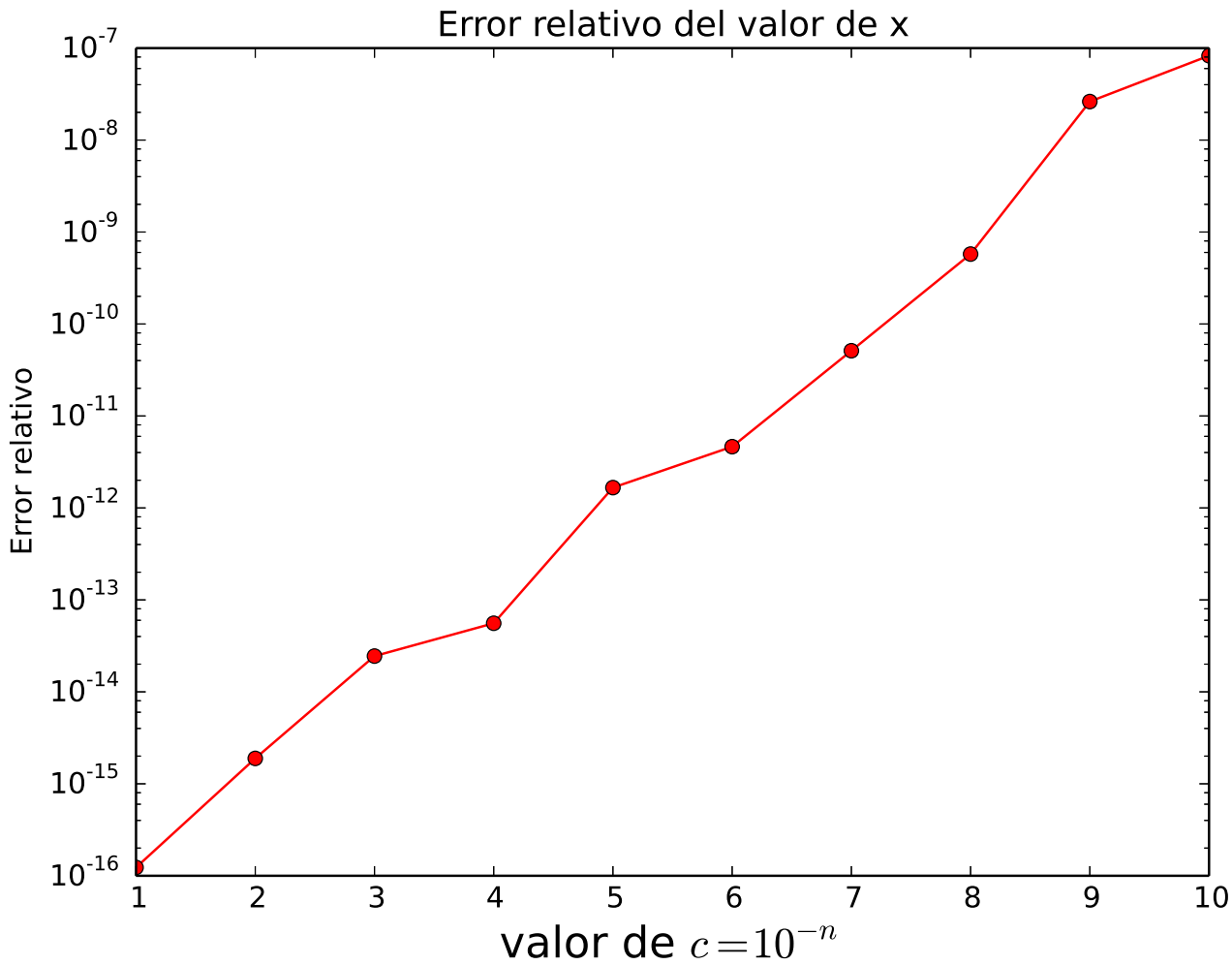


¿Qué estamos viendo?

En la gráfica anterior, vemos una línea muy pegada al eje x, pero hay que considerar que el valor es muy pequeño, mientras que para $c = 10^{-9}$, ya se eleva más el punto y se hace más notorio en la gráfica.

Podríamos pensar que el error relativo es casi el mismo para valores menores de $n = 9$, pero hay que recordar que el ojímetro no funciona bien y menos en física computacional, por lo que ahora hacemos un cambio de escala en el eje y, para ello, usamos la función `semilogy()` de la librería `matplotlib`, que nos cambia la escala a una de tipo semilogarítmico.

Gráfica del error relativo y el valor de c, ajustando el eje y



Tenemos una mejor idea de lo que ocurre

- Vemos que existe una "tendencia" lineal (adviertiendo que uno de los ejes está en escala logarítmica)
- El error relativo va aumentando conforme se incrementa el valor de n.
- Hay algunas variaciones con respecto al valor de error, es decir, la dispersión es notaria, aunque no hemos dicho que debe de ser estrictamente un comportamiento lineal.
- ¿Qué podemos hacer para evitar las operaciones entre valores muy grandes $(b = 1)$ con valores muy pequeños $(4ac)$?