Métodos numéricos

Rodrigo Barrera

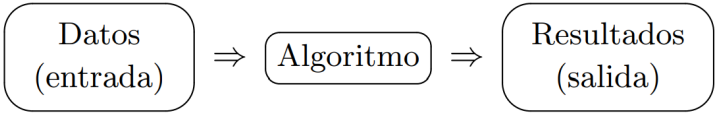
Preliminares

Unidad 1

Evaluaciones

El curso contempla 4 evaluaciones. (2) de índole teórica. (2) de índole computacional.

Las fechas de las evaluaciones serán comentadas a la brevedad.


Datos
(entrada)
⇒
Algoritmo
⇒
Resultados
(salida)

Representación decimal

digitos $= 0, 1, 2, 3, 4, ...9$

Sistema posicional de base 10. Considere el numero 524.503. Este número se representa como sigue;

$$5^2 + 2 \times 10^1 + 4 \times 10^0 + 5 \times 10^{-1} + 0 \times 10^{-2} + 3 \times 10{-}3$$

Números binarios

bits: $\{0, 1\}$

Considere el número 101.011_{10}. Su representación binaria es:

- Parte entera: 101

$$101_{10} = 1100101_2$$

- Parte fraccionaria: 0.011

$$0.011_{10} = 0.000011010_2 \quad \text{(aproximadamente)}$$

Por lo tanto, el número 101.011_{10} en decimal representa aproximadamente el número 1100101.000011010_2 en binario.

$\frac{1}{3}$ en binario

- División: $\frac{1}{3} = 0.3333\ldots$
- Conversión a binario: $0.010101010101\ldots$
  - $0.3333 \times 2 = 0.6666 \rightarrow 0$
  - $0.6666 \times 2 = 1.3332 \rightarrow 1$
  - $0.3332 \times 2 = 0.6664 \rightarrow 0$
  - $0.6664 \times 2 = 1.3328 \rightarrow 1$
  - $0.3328 \times 2 = 0.6656 \rightarrow 0$
  - $0.6656 \times 2 = 1.3312 \rightarrow 1$
- Truncado a seis lugares: 0.010101
- Observación del séptimo lugar: 0
- Redondeo: 0.010101 (hacia abajo)

$\frac{1}{5}$ en binario

- División: $\frac{1}{5} = 0.2$
- Conversión a binario: $0.001100110011\ldots$
  - $0.2 \times 2 = 0.4 \rightarrow 0$
  - $0.4 \times 2 = 0.8 \rightarrow 0$
  - $0.8 \times 2 = 1.6 \rightarrow 1$
  - $0.6 \times 2 = 1.2 \rightarrow 1$
  - $0.2 \times 2 = 0.4 \rightarrow 0$
  - $0.4 \times 2 = 0.8 \rightarrow 0$
- Truncado a seis lugares: 0.001100
- Observación del séptimo lugar: 1
- Redondeo: 0.001101 (hacia arriba)

$\frac{1}{6}$ en binario

- División: $\frac{1}{6} = 0.1666\ldots$
- Conversión a binario: $0.001010101010\ldots$
  - $0.1666 \times 2 = 0.3333 \rightarrow 0$
  - $0.3333 \times 2 = 0.6666 \rightarrow 0$
  - $0.6666 \times 2 = 1.3332 \rightarrow 1$
  - $0.3332 \times 2 = 0.6664 \rightarrow 0$
  - $0.6664 \times 2 = 1.3328 \rightarrow 1$
  - $0.3328 \times 2 = 0.6656 \rightarrow 0$
- Truncado a seis lugares: 0.001010
- Observación del séptimo lugar: 1
- Redondeo: 0.001011 (hacia arriba)

$\frac{1}{7}$ en binario

- División: $\frac{1}{7} = 0.142857\ldots$
- Conversión a binario: $0.001001001001\ldots$
  - $0.142857 \times 2 = 0.285714 \rightarrow 0$
  - $0.285714 \times 2 = 0.571428 \rightarrow 0$
  - $0.571428 \times 2 = 1.142856 \rightarrow 1$
  - $0.142856 \times 2 = 0.285712 \rightarrow 0$
  - $0.285712 \times 2 = 0.571424 \rightarrow 0$
  - $0.571424 \times 2 = 1.142848 \rightarrow 1$
- Truncado a seis lugares: 0.001001
- Observación del séptimo lugar: 0
- Redondeo: 0.001001 (hacia abajo)

$\frac{1}{9}$ en binario

- División: $\frac{1}{9} = 0.1111\ldots$
- Conversión a binario: $0.000111000111\ldots$
  - $0.1111 \times 2 = 0.2222 \rightarrow 0$
  - $0.2222 \times 2 = 0.4444 \rightarrow 0$
  - $0.4444 \times 2 = 0.8888 \rightarrow 0$
  - $0.8888 \times 2 = 1.7776 \rightarrow 1$
  - $0.7776 \times 2 = 1.5552 \rightarrow 1$
  - $0.5552 \times 2 = 1.1104 \rightarrow 1$
- Truncado a seis lugares: 0.000111
- Observación del séptimo lugar: 0
- Redondeo: 0.000111 (hacia abajo)

Doble precisión

Ocho bits forman un byte. La mayoría de los cálculos numéricos se realizan en doble precisión, con números almacenados usando ocho bytes. El formato de un número de doble precisión es:

$$\underbrace{0}_{\text{s}}\ \underbrace{1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11}_{\text{e}}\ \underbrace{12\ldots63}_{\text{f}}$$

$$\# = (-1)^s \times 2^{e-1023} \times 1.f$$

donde s es el bit de signo, e es el exponente sesgado, y $1.f$ es la mantisa. Aquí, e es un número entero escrito en binario, 1023 está en decimal, y f es la parte fraccionaria del número binario después del punto binario.

Distribución de bits

Los 64 bits de un número de doble precisión se distribuyen de la siguiente manera: el signo usa un bit, el exponente usa 11 bits (en decimal, $0 \leq e \leq 2047$), y la mantisa usa 52 bits. La distribución de bits reconcilia dos necesidades conflictivas: que los números deben oscilar entre muy grandes y muy pequeños, y que los números deben estar muy juntos.

Exponentes reservados

Tanto el exponente más grande como el más pequeño están reservados. Cuando e es todo unos ($e = 2047$ en decimal), $f = 0$ se usa para representar el infinito (escrito en Matlab como `Inf`) y $f \neq 0$ se usa para representar "no un número" (escrito en Matlab como `NaN`). `NaN` típicamente resulta de una operación $0/0$, ∞/∞ o $\infty - \infty$. Cuando e es todo ceros, la representación de doble precisión cambia de $1.f$ a $0.f$, permitiendo que estos números denormalizados se reduzcan gradualmente hacia cero. El número positivo de doble precisión más grande es `realmax` $= 1.7977 \times 10^{308}$, y el número normal positivo más pequeño es `realmin` $= 2.2251 \times 10^{-308}$.

Epsilon de máquina

Otro número importante es llamado epsilon de máquina y se define como la distancia entre uno y el siguiente número de máquina más grande. Si $0 \leq \delta < \text{eps}/2$, entonces $1 + \delta = 1$ en aritmética de computadora. Y dado que

$$x + y = x(1 + \frac{y}{x}),$$

cuando $\frac{y}{x} < \text{eps}/2$, entonces $x + y = x$. En doble precisión, el epsilon de máquina es igual a $\text{eps} = 2.2204 \times 10^{-16}$. Nota que el espaciamiento entre números es uniforme entre potencias de 2, pero cambia por un factor de dos con cada potencia adicional de dos. Por ejemplo, el espaciamiento de números entre uno y dos es eps, entre dos y cuatro es $2 \times \text{eps}$, entre cuatro y ocho es $4 \times \text{eps}$, y así sucesivamente.

Ejemplo

A modo de ejemplo, podemos pensar en el problema de hallar $\sqrt{17}$. En este caso, $x = 17$ es el *dato*, f la función raíz cuadrada e $y = \sqrt{17}$ el *resultado* deseado. Como *algoritmo* de cálculo podemos construir, para $\lambda = 17$, la sucesión

$$\begin{cases} x_0 = \lambda \\ x_n = \frac{1}{2}\left(x_{n-1} + \frac{\lambda}{x_{n-1}}\right), \quad n \in \mathbb{N}, \end{cases}$$

Demostración del límite

- Supongamos que la sucesión $\{x_n\}$ converge a un límite L. Entonces,

$$\lim_{n\to\infty} x_n = L.$$

- Sustituimos L en la relación de recurrencia de la sucesión:

$$L = \frac{1}{2}\left(L + \frac{\lambda}{L}\right).$$

- Multiplicamos ambos lados por 2:

$$2L = L + \frac{\lambda}{L}.$$

Demostración del límite

1. Restamos L de ambos lados:
$$L = \frac{\lambda}{L}.$$
2. Multiplicamos ambos lados por L:
$$L^2 = \lambda.$$
3. Tomamos la raíz cuadrada de ambos lados:
$$L = \sqrt{\lambda}.$$

Verificación de L como límite

Hemos encontrado que $L = \sqrt{\lambda}$. Para verificar que este es el límite de la sucesión, observamos que la función de iteración

$$g(x) = \frac{1}{2}\left(x + \frac{\lambda}{x}\right)$$

es contractiva cerca de $\sqrt{\lambda}$. Es decir, si x está cerca de $\sqrt{\lambda}$, $g(x)$ estará aún más cerca de $\sqrt{\lambda}$.

Hemos demostrado que si la sucesión $\{x_n\}$ definida por

$$x_n = \frac{1}{2}\left(x_{n-1} + \frac{\lambda}{x_{n-1}}\right)$$

converge, entonces converge a $\sqrt{\lambda}$. La naturaleza contractiva del método asegura que, dado un punto de partida adecuado, la sucesión convergerá efectivamente a $\sqrt{\lambda}$.

$$\boxed{\lim_{n\to\infty} x_n = \sqrt{\lambda}}$$

- **Signo (S)**: indica si el número es positivo o negativo.
- **Mantisa (M) o Significand**: fracción que representa el dígito significativo del número.
- **Exponente (E)**: indica la potencia de la base a la cual se debe elevar para obtener el valor final del número.

Números máquina

Fórmula general

$$\text{Número} = (-1)^S \times M \times B^E$$

- S: Bit de signo.
- M: Mantisa.
- B: Base (normalmente 2 para sistemas binarios).
- E: Exponente.

Considere el número -123.45:

- **Signo**: $S = 1$ (negativo)
- **Mantisa**: 1.2345 (normalizado)
- **Exponente**: 2 (porque $1.2345 \times 10^2 = 123.45$)

$$-123.45 = (-1)^1 \times 1.2345 \times 10^2$$

Ejemplo en base 2

Considere el número binario -1101.101:

- **Signo**: $S = 1$ (negativo)
- **Mantisa**: 1.101101 (normalizado)
- **Exponente**: 3 (porque $1.101101 \times 2^3 = 1101.101$)

$$-1101.101_2 = (-1)^1 \times 1.101101 \times 2^3$$

Formatos estándar

IEEE 754 define las representaciones de precisión simple y doble:

- **Precisión Simple (32 bits)**:
  - 1 bit para el signo.
  - 8 bits para el exponente (con un sesgo de 127).
  - 23 bits para la mantisa (con un bit implícito).
- **Precisión Doble (64 bits)**:
  - 1 bit para el signo.
  - 11 bits para el exponente (con un sesgo de 1023).
  - 52 bits para la mantisa (con un bit implícito).

Esquema de los errores

En general, los criterios fundamentales para preferir un algoritmo frente a otro son la *rapidez* y la *precisión*; a igualdad de precisión el algoritmo más rápido tendrá, obviamente, la preferencia. El objetivo de, de hecho, el estudio de la precisión o, equivalentemente, del *error*. En muy pocas ocasiones la información de entrada que se suministra es exacta pues se obtiene, en general, mediante instrumentos de medida; como, por otra parte, tanto el almacenamiento de los datos como el propio algoritmo de cálculo introducen también errores, la información de salida contendrá errores que provendrán de las tres fuentes. Esquemáticamente:

Errores en métodos numéricos

El error en la resolución de un problema de ingeniería o ciencia puede surgir debido a varios factores:

- **Errores en la técnica de modelado:**
  - Un modelo matemático puede basarse en suposiciones que no son aceptables.
  - Por ejemplo, suponer que la fuerza de arrastre en un automóvil es proporcional a la velocidad del automóvil, cuando en realidad es proporcional al cuadrado de la velocidad, puede introducir errores significativos.
- **Errores en la implementación y medición:**
  - Los errores pueden surgir de fallos en los programas o en la medición de cantidades físicas.

Errores en métodos numéricos (cont.)

Dentro del contexto de los métodos numéricos, los dos tipos de errores principales a considerar son:

1. **Error de redondeo:**
   - Ocurre debido a las limitaciones en la representación de números en la computadora, donde solo se pueden almacenar un número finito de cifras significativas.
2. **Error de truncamiento:**
   - Surge cuando se utilizan aproximaciones para representar operaciones matemáticas continuas, como cuando se corta una serie infinita después de un número finito de términos.

Midiendo el error

Supongamos que tenemos un número a y una aproximación $\tilde{a}$. Vamos a considerar dos formas de medir el error en dicha aproximación.

Midiendo el error

La primera medida de error es la más obvia, es esencialmente la diferencia entre a y $\tilde{a}$.

Definición (error absoluto). Supongamos que $\tilde{a}$ es una aproximación al número a. El error absoluto en esta aproximación está dado por $|a - \tilde{a}|$.

Si $a = 100$ y $\tilde{a} = 100.1$, el error absoluto es 0.1, mientras que si $a = 1$ y $\tilde{a} = 1.1$, el error absoluto sigue siendo 0.1. Nótese que si a es una aproximación a $\tilde{a}$, entonces $\tilde{a}$ es una aproximación igualmente buena a a con respecto al error absoluto.

Midiendo el error

Al realizar operaciones aritméticas, los errores de redondeo se propagan. Por ejemplo, el error relativo en la suma de x e y puede aproximarse como:

$$\frac{|x+y-\hat{x}-\hat{y}|}{|x+y|} \approx \frac{|x|}{|x+y|}\epsilon_x + \frac{|y|}{|x+y|}\epsilon_y$$

Error relativo en la suma

```
def error_relativo_suma(x, y, epsilon_x, epsilon_y):

    # Calcular el valor absoluto de x e y
    abs_x = abs(x)
    abs_y = abs(y)
    abs_sum = abs_x + abs_y

    # Calcular el error relativo en la suma
    error_relativo = (abs_x / abs_sum) * epsilon_x + (abs_y / abs_sum) *
        epsilon_y

    return error_relativo
```

Error relativo recíproco

Supongamos que $\tilde{a}$ es una aproximación de a con un error relativo $r < 1$. Entonces a es una aproximación de $\tilde{a}$ con un error relativo

$$\frac{|a - \tilde{a}|}{|\tilde{a}|} \leq \frac{r}{1 - r}.$$

Demostración (1)

Si $\tilde{a}$ es una aproximación de a con un error relativo r, entonces:

$$\frac{|a - \tilde{a}|}{|a|} \leq r.$$

Queremos encontrar el error relativo de a como aproximación de $\tilde{a}$:

$$\frac{|a - \tilde{a}|}{|\tilde{a}|}.$$

Demostración (2)

Partimos de la desigualdad inicial:

$$|a - \tilde{a}| \leq r|a|.$$

Sumamos $|\tilde{a}|$ a ambos lados:

$$|a| - r|a| \leq |\tilde{a}|,$$

lo que simplifica a:

$$|a|(1 - r) \leq |\tilde{a}|.$$

Demostración (3)

Despejamos $|a|$:

$$|a| \leq \frac{|\tilde{a}|}{1-r}.$$

Consideramos el error relativo recíproco:

$$\frac{|a-\tilde{a}|}{|\tilde{a}|}.$$

Demostración (4)

Sabemos que:

$$|a - \tilde{a}| \leq r|a|.$$

Sustituyendo la cota de $|a|$:

$$|a - \tilde{a}| \leq r\left(\frac{|\tilde{a}|}{1-r}\right).$$

Demostración (5)

Dividiendo ambos lados por $|\tilde{a}|$, obtenemos:

$$\frac{|a - \tilde{a}|}{|\tilde{a}|} \leq \frac{r}{1 - r}.$$

Por lo tanto, hemos demostrado que:

$$\frac{|a - \tilde{a}|}{|\tilde{a}|} \leq \frac{r}{1 - r}.$$

Error relativo recíproco

Primero observamos que, dado que

$$1 > r = \frac{|a - \tilde{a}|}{|a|} = \left|1 - \frac{\tilde{a}}{a}\right|,$$

ambos a y $\tilde{a}$ deben tener el mismo signo, pues de lo contrario el lado derecho de esta desigualdad sería mayor que 1. Por otro lado, también tenemos a partir de la desigualdad triangular que

$$1 = \left|1 - \frac{\tilde{a}}{a} + \frac{\tilde{a}}{a}\right| \leq \left|1 - \frac{\tilde{a}}{a}\right| + \left|\frac{\tilde{a}}{a}\right|$$

Aritmética de punto flotante

Un número en punto flotante x se puede expresar en la forma normalizada:

$$x = (-1)^s \cdot m \cdot 2^e$$

donde:

- s es el bit de signo (0 para positivo, 1 para negativo),
- m es la mantisa o significando, con un valor en el intervalo $[1, 2)$ para números normalizados,
- e es el exponente.

Ejemplo 1: Número positivo 6.5

- Parte entera: Convertimos 6 a binario.

$$6_{10} = 110_2$$

- Parte fraccionaria: Convertimos 0.5 a binario.

$$0.5_{10} = 0.1_2$$

Ejemplo 1: Continuación

- Juntamos ambas partes:

$$6.5_{10} = 110.1_2$$

- Normalizamos:

$$110.1_2 = 1.101_2 \times 2^2$$

- En punto flotante: $s = 0$, $m = 1.101$, $e = 2$.

Así, 6.5 en punto flotante se representa como:

$$x = (-1)^0 \cdot 1.101 \cdot 2^2$$

Aritmética de punto flotante

Los números de punto flotante no pueden representar todos los números reales exactamente, lo que introduce errores de redondeo. Si $\hat{x}$ es la representación en punto flotante de x, el error absoluto de redondeo es:

$$|x - \hat{x}|$$

La precisión relativa está dada por:

$$\text{Precisión relativa} = 2^{-p}$$

Operaciones aritméticas en coma flotante

De hecho, la aritmética estándar en coma flotante está diseñada mediante operaciones $\oplus, \ominus, \otimes, \oslash$ que verifican, para cada par de números máquina x e y:

$$
\begin{aligned}
x \oplus y &= r(x + y), \\
x \ominus y &= r(x - y), \\
x \otimes y &= r(x \times y), \\
x \oslash y &= r(x / y).
\end{aligned}
$$

La implementación efectiva de las operaciones es bastante sofisticada, pues debe enfrentarse con diversos problemas.

Pasos para sumar o restar dos números máquina

1. Se igualan los exponentes al mayor de ambos.
2. Se realiza la operación de forma exacta.
3. Se normaliza el resultado (modificando, si es necesario, el exponente de forma que la mantisa esté entre 1 y 2, es decir, de la forma $\pm 1.m \times 2^E$).
4. Se redondea (en caso de que sea necesario).
5. Se normaliza si es necesario.

Ejemplo 1: Suma de números máquina

Para sumar $1 \equiv 1.0 \times 2^0$ con $2 \equiv 1.0 \times 2^1$, se escribe:

$$\begin{aligned} & 0.10\underbrace{00\ldots0}_{22} \times 2^1 \\ +\, & 1.00\underbrace{00\ldots0}_{22} \times 2^1 \\ = \, & 1.10\underbrace{00\ldots0}_{22} \times 2^1 \end{aligned}$$

Como se observa, solo han hecho falta los dos primeros pasos.

Ejemplo 1: Resta de números máquina

Si restamos a $1 \equiv 1.0 \times 2^0$ el número máquina más cercano a él por la izquierda, es decir, $1.1\underbrace{00\ldots1}_{23} \times 2^{-1}$, tenemos:

$$
\begin{aligned}
& 1.0\underbrace{00\ldots0}_{23} \times 2^0 \\
-\, & 0.1\underbrace{00\ldots1}_{23} \times 2^0 \\
= \, & 0.0\underbrace{00\ldots1}_{23} \times 2^0
\end{aligned}
$$

Para realizar la operación de forma exacta hemos tenido que utilizar una posición más de memoria (24 bits) para guardar el segundo número y el resultado.

Multiplicación y división de números máquina

Para la multiplicación (o división) de dos números no es necesario igualar los exponentes; basta multiplicar las dos mantisas (lo que dará un número de, a lo más, 48 cifras significativas, que cabe en un registro) y se suman los exponentes, normalizando el resultado, redondeándolo si es preciso y, eventualmente, volviéndolo a normalizar.

Fenómenos en aritmética de coma flotante

- Las operaciones en coma flotante no verifican, en general, la propiedad asociativa.
- Si dos números máquina x e y verifican $x \oplus y = x$, esto no implica, en general, que $y = 0$.
- La cancelación aparece al restar dos números muy próximos entre sí, lo que puede producir grandes errores relativos.

Ejemplo 2: Cancelación

Al realizar la resta en el ordenador de los números:

$$x = 1.1\underbrace{00\ldots1}_{23}\underbrace{00\ldots1}_{15}\times 2^E,$$
$$y = 1.1\underbrace{00\ldots1}_{15}\times 2^E,$$

(teniendo en cuenta que y es ya un número máquina) se obtiene:

$$x \ominus y = r(r(x) - y) = r(0.0\underbrace{00\ldots1}_{15}\underbrace{00\ldots1}_{8}\times 2^E) = 1.1\underbrace{00\ldots1}_{7}\times 2^{E-16}.$$

Propagación del error

A la hora de resolver un problema en un ordenador, los datos de partida no tienen por qué ser exactamente los datos originales debido al redondeo. Además, aunque los datos se almacenaran de forma exacta, en el momento en que empezamos a trabajar con ellos se cometen errores, y el resultado obtenido puede distar mucho del deseado.

Ejemplo: Suma de dos números

Analicemos la suma de dos números $x, y \in \mathbb{R}$:

$$\begin{aligned} x \oplus y &= r(r(x) + r(y)) \\ &= r((1+\delta_1)x + (1+\delta_2)y) \\ &= (1+\delta_3)[(1+\delta_1)x + (1+\delta_2)y] \\ &= (1+\delta_3)(x+y) + (1+\delta_3)\delta_1 x + (1+\delta_3)\delta_2 y \end{aligned}$$

con $|\delta_i| \leq 2^{-24} = \frac{eps}{2}$ para $i = 1, 2, 3$. Los errores de redondeo en los datos se propagan al resultado de la operación.

Propagación de errores

Si el resultado es un operando de otra operación, este error se propaga al consiguiente resultado, y así sucesivamente. En un algoritmo que involucre muchas operaciones elementales, los errores se acumulan.

- Para cualquier operación elemental $(+, -, \cdot, /)$:

$$r(x * y) = (1 + \delta)(x * y) \text{ con } |\delta| \leq \frac{eps}{2}.$$

- Si x e y no son números máquina:

$$r(r(x) * r(y)) = r([(1 + \delta_1)x] * [(1 + \delta_2)]y) = (1 + \delta_3)((1 + \delta_1)x * (1 + \delta_2)y).$$

Operaciones compuestas

Supongamos que queremos calcular $x(y+z)$ donde x, y y z son números máquina:

$$\begin{aligned} x \otimes (y \oplus z) &= r(xr(y+z)) \\ &= (1+\delta_1)(xr(y+z)) \\ &= (1+\delta_1)[(1+\delta_2)x(y+z)] \\ &= (1+\delta_1+\delta_2+\delta_1\delta_2)x(y+z) \\ &\approx (1+\delta_1+\delta_2)x(y+z) \\ &= (1+\delta_3)x(y+z) \end{aligned}$$

con $|\delta_3| \leq |\delta_1| + |\delta_2| \leq 2^{-24} + 2^{-24} = 2^{-23}$.

Importancia del estudio de la propagación del error

Debemos estudiar cuánto influye la propagación del error en el resultado final del problema.
Dos conceptos principales están ligados a este estudio:

- **Condicionamiento**: Mide la influencia que tendrían en el resultado eventuales errores en los datos, en el caso ideal de que se pudiese trabajar con aritmética exacta; está ligado al problema en sí y no depende del algoritmo.
- **Estabilidad**: Relacionada con la influencia que tiene en los resultados finales la acumulación de los errores que se producen en las sucesivas operaciones elementales que se llevan a cabo para resolver el problema; depende del algoritmo utilizado para obtener la solución.

Propagación del error

```
direct_geometric_sum <- function(a, r, n) {
  sum <- 0
  for (i in 0:(n-1)) {
    sum <- sum + a * r^i
  }
  return(sum)
}
# Funcion para calcular la suma de una serie geometrica usando la formula
    cerrada
closed_form_geometric_sum <- function(a, r, n) {
  if (r == 1) {
    return(a * n)
  } else {
    return(a * (1 - r^n) / (1 - r))
```

Un poco de historia

El 4 de junio de 1996, el cohete Ariane 5, desarrollado por la Agencia Espacial Europea (ESA), fue lanzado en su vuelo inaugural. Sin embargo, a los 37 segundos del despegue, el cohete se desvió de su trayectoria y se autodestruyó. Este evento fue causado por un fallo en el software de control del cohete, relacionado con la propagación del error numérico.

En el software del Ariane 5, se realizó una conversión de un número entero de 64 bits a un número de punto flotante de 16 bits. El valor del número entero era demasiado grande para ser representado en el formato de punto flotante de 16 bits, lo que causó un desbordamiento.

Un poco de historia

El 25 de febrero de 1991, un misil Scud iraquí impactó en un cuartel estadounidense en Dhahran, Arabia Saudita, matando a 28 soldados. El sistema de defensa de misiles Patriot, diseñado para interceptar misiles enemigos, falló en detectar y destruir el misil Scud. La causa del fallo fue un error de precisión en el cálculo del tiempo debido a la propagación de errores numéricos.

El error fue debido a la conversión de tiempo de 24 bits a una representación de punto flotante de 24 bits. La computadora del sistema Patriot utilizaba un reloj interno que contaba el tiempo en intervalos de 0.1 segundos, almacenando estos valores en un número de punto flotante de 24 bits. Este formato no podía representar con precisión todos los números necesarios, lo que resultó en una acumulación de errores a lo largo del tiempo.

Ejercicio 1

El coeficiente binomial $\binom{n}{i}$ se define como

$$\binom{n}{i} = \frac{n!}{i!(n-i)!} \tag{1}$$

donde $n \geq 0$ es un número entero e i es un número entero en el intervalo $0 \leq i \leq n$. Los coeficientes binomiales aparecen en varios contextos y a menudo deben ser calculados en una computadora. Dado que todos los coeficientes binomiales son enteros (esto significa que la división en 1 nunca puede dar un residuo), es razonable usar variables enteras en tales cálculos. Para valores pequeños de n e i, esto funciona bien, pero para valores mayores podemos encontrarnos con problemas porque el numerador y el denominador en 1 pueden volverse más grandes que el entero máximo permitido, incluso si el coeficiente binomial en sí mismo puede ser relativamente pequeño.

Ejercicio 1

Una característica desafortunada de la fórmula 1 es que, incluso si el coeficiente binomial es pequeño, el numerador y el denominador pueden ser grandes. En general, esto es malo para los cálculos numéricos y debe evitarse si es posible. Si consideramos la fórmula 1 con más detalle, notamos que muchos de los números se cancelan,

$$\binom{n}{i} = \frac{1 \cdot 2 \cdots i \cdot (i+1) \cdots n}{1 \cdot 2 \cdots i \cdot 1 \cdot 2 \cdots (n-i)} = \frac{i+1}{1} \cdot \frac{i+2}{2} \cdots \frac{n}{n-i}. \tag{2}$$

Ejercicio 1

¿Es posible encontrar números demasiado grandes durante esos cálculos si el coeficiente binomial a calcular es más pequeño que el número en punto flotante más grande que puede ser representado por tu computadora?

Ejercicio 2: Propagación del error en una suma repetitiva

Realiza la suma repetitiva de un número muy pequeño muchas veces y compara el resultado con el valor esperado teórico.

Instrucciones:

1. Define un número muy pequeño, por ejemplo, $\epsilon = 1 \times 10^{-10}$.
2. Suma ϵ un millón de veces.
3. Compara el resultado de la suma con el valor teórico esperado (1).

Ejercicio 3: Propagación del error en una serie geométrica

Calcula la suma de una serie geométrica utilizando dos métodos diferentes y compara los resultados para observar la propagación del error.

Instrucciones:

1. Define el primer término $a = 1$, la razón $r = 0.9999999$, y el número de términos $n = 1,000,000$.
2. Calcula la suma de la serie geométrica directamente.
3. Calcula la suma de la serie geométrica usando la fórmula cerrada.
4. Compara los resultados obtenidos por ambos métodos.

Ejercicio 4: Suma y resta de números en coma flotante

Realiza la suma y resta de dos números en coma flotante y analiza los errores de redondeo que ocurren.

Instrucciones:

1. Define dos números en coma flotante $x = 1.234567 \times 10^5$ y $y = 1.234568 \times 10^5$.
2. Calcula la suma $s = x + y$.
3. Calcula la resta $d = x - y$.
4. Analiza los resultados obtenidos considerando los errores de redondeo.

Ejercicio 5: Propagación del error en operaciones aritméticas

Observa la propagación del error al realizar una serie de operaciones aritméticas en coma flotante.

Instrucciones:

1. Define un número muy pequeño $\epsilon = 1 \times 10^{-10}$.
2. Realiza la operación $\mathit{result} = (1 + \epsilon) - 1$.
3. Compara el resultado con el valor teórico esperado (ϵ).
4. Discute cómo el error de redondeo afecta el resultado final.

Ejercicio 6: Multiplicación y división de números en coma flotante

Realiza la multiplicación y división de dos números en coma flotante y analiza los errores de redondeo que ocurren.

Instrucciones:

1. Define dos números en coma flotante $a = 1.234567 \times 10^5$ y $b = 1.234568 \times 10^{-5}$.
2. Calcula el producto $p = a \times b$.
3. Calcula el cociente $q = a/b$.
4. Analiza los resultados obtenidos considerando los errores de redondeo.