Análisis de error

Análisis Numérico

Joaquin Cavieres G.

Error de analisis

Representación binaria y decimal

Cada número real tiene una representación binaria y una decimal. Algunas veces podemos encontrar la palabra "representación" como tambíen la palabra "expansión" en la literatura para estas representaciones en el computador.

I) Representación de números enteros

$$(71)_{10} = 7 \times 10^1 + 1 \times 10^0$$

y su equivalente en número binario:

$$\begin{aligned}(1000111)_2 =& 1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 \\ &+ 1 \times 2^1 + 1 \times 2^0\end{aligned}$$

Definición 1:

Un número real decimal de punto flotante distinto de 0 "x" tiene la siguiente representación:

$$x = \sigma \times (\bar{x})_{10} \times 10^n,$$

donde $\sigma = \pm 1$ es el signo,$\bar{x}$ es la mantisa y n es el exponente.

Definición 2:

Un número real decimal de punto flotante binario distinto de 0 "x" tiene la siguiente representación:

$$x = \sigma \times (\bar{x})_2 \times 10^e,$$

donde $\sigma = \pm 1$ es el signo,$\bar{x}$ es la mantisa y e es el exponente.

Estas representaciones se dicen que estan normalizadas si:

- En el caso de números decimales la mantisa satisface que $(1)_{10} \leq \bar{x} < (10)_{10}$
- En el caso de números binarios la mantisa satisface que $(1)_2 \leq \bar{x} < (10)_2$

Los dígitos significativos de un número son los dígitos de la mantisa sin contar los ceros iniciales. Por lo tanto, para los números normalizados, el número de dígitos significativos es el mismo que el número de dígitos de la mantisa. La precisión de una representación es el número máximo, p, de dígitos significativos que se pueden representar. Para una representación normalizada, la precisión coincide con el número de dígitos de la mantisa.

La precision puede ser finita si $p < \infty$ o infinita si no hay un límite al número de dígitos en la mantisa.

Ejemplo 1

- Considere el número $x = 314{,}15$, en donde su normalización decimal de número de punto flotante es:

$$\sigma = \pm 1, \;\; \bar{x} = 3{,}1415, \;\; , n = 2$$

Como se puede ver la representación tiene 5 digítos significantes.

- El número binario $x = (101101{,}11001)_2$ tiene una forma normalizada representada por $(1{,}010111001)_2 \times 2^4$ con 10 digítos significantes.
- El número $x = (101{,}001101)_2 = (5{,}203125)_{10}$ tiene un número flotante decimal normalizado:

$$\sigma = \pm 1, \;\; \bar{x} = 5{,}203125, \;\; , n = 0$$

mientras que la normalización binaria de punto flotante es:

$$\sigma = \pm(1)_2, \;\; \bar{x} = (1{,}01001101)_2, \;\; , e = (2)_{10} = (10)_2$$

por tanto el número tiene 7 digítos sinigificativos para una representación decimal y 9 para la representación binaria.

Ejemplo 2

Supongamos que para una representación binaria tenemos p digítos en la mantisa. Si la representación de un número dado x puede ser normalizada, entonces esta debe tener la forma:

$$x = \pm 1.b_1 b_2 .... b_{p-1} \times 2^e$$

Ya que esta representación no puede tener ceros en la izquierda, la precisión de esta representación es p. Ahora, supongamos que la representación de x no puede ser normalizada, así tiene la siguiente forma:

$$x = \pm 0{,}0 \ldots, 0 b_j \ldots b_{p-1} \times 2^e$$

donde $b \neq 0$ y $j \leq p - 1$. La precisión entonces de esa representación es $p - j$.

Conversión de decimal a binario (y viceversa)

Desde un número binario a un decimal es sencillo, por ejemplo:

$$(1101011)_2 = 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

Visto de otra manera tenemos:

2^6	2^5	2^4	2^3	2^2	2^1	2^0
64	32	16	8	4	2	1
1	1	0	1	0	1	1

y sumamos donde tenemos valores 1 (del número binario) en la tabla:

$$64 + 32 + 8 + 2 + 1 = 107$$

107 es la representación decimal del número binario $(1101011)_2$.

Para convertir de decimal a binario vamos a considerar al número 50 como ejemplo y la tabla de número binario en base 2. Por lo tanto, la conversión se realiza de la siguiente manera:

- Revisamos si el número elegido (50 en este caso) es mayor o igual que el número de la tabla de IZQUIERDA A DERECHA.
- Si es mayor entonces restamos al número dado el número de esa posición y así sucesivamente.

2^6	2^5	2^4	2^3	2^2	2^1	2^0
64	32	16	8	4	2	1
0	1 50 - 32 = 18	1 18 - 16 = 2	0	0	1 2 - 2 = 0	0

Ejemplos de números "grandes en R:

```
num_grande = 35000000000
num_grande
```

```
## [1] 3.5e+10
```

```
# Que no lo muestre en notación científica
format(num_grande, scientific = FALSE)
```

```
## [1] "35000000000"
```

```
# Cambiar "," por ".", etc
format(num_grande, scientific = FALSE, big.mark = ',')
```

```
## [1] "35,000,000,000"
```

Aritmética de dígitos finitos

Ya que existe una representación inexacta de números en el computador, la arítmetica del computador tampoco lo es, puesto que las operaciones de digítos binarios necesita realizar operaciones de cambio y lógicas.

Considere la representación de punto flotante $fl(x)$ y $fl(y)$ para los números x e y. Los símbolos $\oplus$, $\ominus$, $\otimes$, $\oslash$, representan operaciones de máquina como la suma, resta, multiplicación y división, respectivamente. De lo anterior podemos realizar una aritmética de dígitos finitos dada por:

$$\begin{aligned} x \oplus y &= fl(fl(x) + fl(y)), \quad x \otimes y = fl(fl(x) * fl(y)) \\ x \ominus y &= fl(fl(x) - fl(y)), \quad x \oslash y = fl(fl(x)/fl(y)) \end{aligned}$$

Ejemplo 3

Supongamos que $x = \frac{5}{7}$ e $y = \frac{1}{3}$. Mediante el método de corte de 5 dígitos calculamos $x + y$ y $x * y$.

Solución:

$$\begin{aligned} x =& \frac{5}{7} = 0{,}7142857 \rightarrow 0.\overline{714285} \\ y =& \frac{1}{3} = 0{,}3333333 \rightarrow 0.\bar{3} \end{aligned}$$

Lo que implica que los valores de corte de 5 dígitos para x e y son:

$$\begin{aligned} fl(x) =& 0{,}71428 \times 10^0 \\ fl(y) =& 0{,}33333 \times 10^0 \end{aligned}$$

así podemos hacer para $x + y$:

$$\begin{aligned} x \oplus y =& fl(fl(x) + fl(y)) = fl(0{,}71428 \times 10^0 + 0{,}33333 \times 10^0) \\ =& fl(1{,}04761 \times 10^0) = 0{,}10476 \times 10^1 \end{aligned}$$

El resultado de $x + y$ fuera de la máquina (computador sería:

$$x + y = \frac{5}{7} + \frac{1}{3} = \frac{22}{21}$$

lo que finalmente nos daría como resultado:

$$\text{Error absoluto} = \left|\frac{22}{21} - 0{,}10476 \times 10^{1}\right| = 0{,}190 \times 10^{-4}$$

y

$$\text{Error relativo} = \left|\frac{0{,}190 \times 10^{-4}}{\frac{22}{21}}\right| = 0{,}182 \times 10^{-4}$$

Haga usted $x * y$ lo que le debe dar como resultado: $0{,}23809 \times 10^{0}$, valor real $= 5/21$, error absoluto $= 0{,}524 \times 10^{-5}$, error relativo $= 0{,}220 \times 10^{-4}$

Ejemplo 4

Suponga que ademas de $x = \frac{5}{7}$ e $y = \frac{1}{3}$ tenemos:

$$\begin{aligned} u &= 0{,}714251 \\ v &= 98765{,}9 \\ w &= 0{,}111111 \times 10^{-4} \end{aligned}$$

por lo que tenemos los siguientes números de punto flotante:

$$\begin{aligned} fl(u) &= 0{,}71425 \times 10^{0} \\ fl(v) &= 0{,}98765 \times 10^{5} \\ fl(w) &= 0{,}11111 \times 10^{-4} \end{aligned}$$

Calcular los valores con el método de corte para 5 dígitos para $x \ominus u$ y $(x \ominus u) \oslash w$.

Solución:

En estas operaciones se puede ver algunos problemas que se pueden presentar al realizar aritmética de dígitos finitos. Ya que x y u son muy parecidos entre sí, el error absoluto para $x \ominus u$ sería:

$$\begin{aligned} |(x-u) - (x \ominus u)| &= |(x-u) - (fl(fl(x) - fl(u)))| \\ &= \left|\left(\frac{5}{7} - 0{,}714251\right) - fl(0{,}71428 \times 10^{0} - 0{,}71425 \times 10^{0})\right| \\ &= |0{,}347143 \times 10^{-4} - fl(0{,}00003 \times 10^{0})| = 0{,}47143 \times 10^{-5} \end{aligned}$$

pero su error relativo grande:

$$\left|\frac{0{,}47143 \times 10^{-5}}{0{,}347143 \times 10^{-4}}\right| \leq 0{,}136$$

Desbordamiento (Overflow) y subdesbordamiento (Underflow)

Ya que en `R` los tipos de datos numéricos pueden contener un amplio rango de números es poco probable que el desbordamiento y el subdesbordamiento se produzcan como en otros sistemas. Sin embargo, vamos a comprender este tipo de problemas cuando `R` interactua con otros entornos operativos.

El subdesbordamiento se produce cuando tenemos un número demasiado pequeño como para ser representado por el computador como distinto de 0 (independiente del signo del número). Los tipos de datos numéricos de precisión doble (*double*) pueden representar números tan pequeños que el número no se puede utilizar de manera realista en la mayoría de las operaciones.

```
.Machine$double.xmin
```

```
## [1] 2.225074e-308
```

Este número, si se imprime directamente, sería un decimal seguido de casi 300 números 0's antes de que apareciera el primer número distinto de cero. Este número es significativamente más pequeño que casi todos los números que se pueden usar en la práctica. Estos valores pueden llegar a existir y se encuentran entre 0 y `.Machine$double.xmin`.. Mostrar un ejemplo práctico o realista es complicado pero nos podríamos poner en este escenario si dividimos `.Machine$double.xmin` por dos:

```
.Machine$double.xmin / 2
```

```
## [1] 1.112537e-308
```

En este caso el computador al parecer tiene un métoodo interno para almacenar números incluso más pequeños que `.Machine$double.xmin`. para una representación de un número de punto flotante (al parecer moviendo los ceros iniciales a la mantisa). Estos números son llamados *desnormalizados* y llevan al límite el subdesbordamiento incluso más bajo que `.Machine$double.xmin`. Lo anterior es una representación que depende de la máquina y probablemente no estar disponible.

En el otro extremo, cuando tenemos números demasiado grandes para ser expresados, entonces se produce el desbordamiento. Como el subdesbordamiento, el desbordamiento es un problema teórico pero en terminos practicos se puede crear un ejemplo de el. En `R` podemos llamar al número más grande mediante:

```
.Machine$double.xmax
```

```
## [1] 1.797693e+308
```

Ya que los mismos bits con exponentes negativos pueden contener exponentes positivos, este número está seguido por más de 300 ceros, así este número es significativamente mayor que casi todos los números que podrían usarse en la práctica. Para tener unea idea, los cosmologos estiman que hay 10^{80} atómos de hidrógeno en TODO el Universo.

Al igual que los números más pequeños, incluso en los límites de las matemáticas o la física, hay números que son mucho más pequeños que el límite superior de los números de presición doble.

Como ejemplo nos concentraremos en los números almacenados como enteros y, al duplicarlos, el valor resultante sea un entero con desbordamiento:

```
2147483647L * 2L
```

```
## [1] NA
```

Vea que al especificar la letra "L" estamos obligando a `R` a trabajar con un número entero.

Una ventaja de utilizar los tipos de datos de precisión doble para almacenar números es que pueden almacenar enteros hasta 2^{53} sin ningun tipo de pérdida de precisión.

Tanto en el caso de subdesbordamiento como de desbordamiento, `R` no tiene problemas de error ya que simplemente usa la mejor opción disponible. En el caso de subdesbordamiento, usar 0 probablemente sea una respuesta suficientemente buena. En el caso de desbordamiento, es poco probable que infinito sea una respuesta suficientemente buena, pero si ocurre, entonces se debería indicar adecuadamente que no se está en el camino correcto.

Error de propagación y estabilidad

Estos tipos de errores se pueden presentar en cualquier momento, sin embargo, en las operaciones generales estos tipos de errores probalemente no sean de relativa importancia. No obstante nosotros generalmente utilizamos los computadores para resolver provlemas complejos que requieren muchos cálculos matemáticos.

Como por ejemplo, calculos en matrices, requieren numerosos pasos sucesivos para llegar a un resultado correcto. Cada una de esas operaciones está sujeta a errores numéricos y esos errores pueden agravarse en forma de error en cascada. Comprender cómo los errores en un cálculo afectan a otros cálculos en una operación compleja se conoce como error de propagación.

Referencias

Howard, J. P. (2017). Computational Methods for Numerical Analysis with R. CRC Press.

Burden, R. L., & Faires, J. D. (2011). Numerical analysis.