Ejercicios

Análisis Numérico

Joaquin Cavieres G.

Error de análisis

Cuando resolvemos un problema mediante algún método numérico siempre es importante conocer los tipos de errores que se pueden producir en nuestra aproximación. Por lo tanto, es necesario considerar todas las fuentes de errores asociadas con el problema para así buscar formas de reducir sus efectos en los resultados.

Dentro del análisis numérico el estudio de estos errores debe ser uno de los principales objetivos a considerar, y para el computo matemático y/o estadística, necesariamente debe ser el mínimo error posible. De esta manera podemos integrar dentro del proceso de análisis (incluyendo los resultados) la reducción del error asociado al desarrollo de los modelos matemáticos.

Fuentes de error

Formulación del modelo matemático

- Número de datos
- Incertidumbre asociada a los datos
- Planteamiento del problema

Formulación del modelo discretizado

- Tipo de modelo discretizado
- Aproximación de datos y variables
- El problema está mal condicionado

Selección de métodos resolutivos

- Errores en algoritmos
- Errores en máquinas

Tipos de error

Vamos a dar una breve descripción de los tipos de errores que comúnmente podemos encontrar.

Supongamos que $p^\star$ es una aproximación de p. De lo anterior podemos definir a:

- error real $= p - p^\star$
- error absoluto $= |p - p^\star|$
- error relativo $= \frac{|p-p^\star|}{|p|}$, siempre y cuando $p \neq 0$.

Generalmente no conocemos el valor real de p, pero si es posible asignar límites $\epsilon_{p^\star}$ y $\alpha_{p^\star}$:

$$|p - p^\star| \leq \epsilon_{p^\star}, \quad \frac{|p - p^\star|}{|p^\star|} \leq \alpha_{p^\star}$$

por lo que podemos escribir a p como:

$$p = p^\star \pm \epsilon_{p^\star}$$

o

$$p = p^\star(1 \pm \alpha_{p^\star})$$

ya que $|p - p^\star| \leq \alpha_{p^\star}|p^\star|$.

Ejemplo 1

Si consideramos a $p = 2$ y a $p^\star = 2{,}1$, el error absoluto es igual a 0.1 y el error relativo es a 0.05. Por otra parte tenemos que $p = 2000$ y a $p^\star = 2100$. En este último caso el error absoluto es 100 y el error relativo es 0.05.

Esto nos indica que mediante las mismas aproximaciones tenemos el mismo error relativo pero el error absoluto es diametralmente grande entre ambos ejemplos.

Para medir el error generalmente se usa el error relativo ya que el error absoluto puede llevar a malas intepretaciones o confusiones. El error relativo es más apropiado en estos casos ya que considera la magnitud de valor el cual estamos midiendo.

Para dar una referencia a los límites de los errores aboslutos y relativos podemos definir a:

Aproximación mediante truncamiento (método de corte)

- $p^\star$ se aproxima a p con d dígitos decimales si d es el mayor entero positivo para que se cumpla:

$$|p - p^\star| \leq 10^{-d}$$

- $p^\star$ se aproxima a p con c cifras significativas si c es el mayor entero positivo para que se cumpla:

$$\frac{|p - p^\star|}{|p|} \leq 10^{1-c}$$

Aproximación mediante redondeo

- $p^\star$ se aproxima a p con d dígitos decimales si d es el mayor entero positivo para que se cumpla:

$$|p - p^\star| \leq 0{,}5 \times 10^{-d}$$

- $p^\star$ se aproxima a p con c cifras significativas si c es el mayor entero positivo para que se cumpla:

$$\frac{|p - p^\star|}{|p|} \leq 0{,}5 \times 10^{1-c} = 5 \times 10^{-c}$$

Ejemplo 2

Para p = 55.47846, calcule el número mediante redondeo a 4 digítos decimales. Además, calcule el error de análisis.

Si consideramos 4 digítos decimales entonces:

$$\begin{aligned} y_4 &= 55{,}47846 \\ |y - y_4| &= |55{,}47846 - 55{,}47845| = 1{,}0 \times 10^{-5} \leq \frac{1}{2} \times 10^{-4} \end{aligned}$$

Números de punto flotante

Los computadores tienen la funcionalidad para los números enteros y de punto flotante (o de máquina) para representar números reales (números enteros y números reales respectivamente). Para representar un número de punto flotante se permiten números de gran tamaño pero con limitaciones de magnitud como en el número de dígitos. La representación de punto flotante está relacionada específicamente con la notación científica, la cual facilita la representación de este tipo de números reales con una cierta precisión para así simplificar las operaciones aritméticas usuales que con ellos se realizan.

Sistemas de números posicionales

Los números reales generalmente son representados mediante el sistema decimal de base 10 (sistema posicional). Mediante esta representación los números reales tienen una estructura de diez caracteres (digítos decimales) y son el 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9. La **magnitud** de un digíto u_i que representa al número real depende de su posición en u_i.

Representación de punto flotante

Considere el siguiente número:

$$0 \;\; 10000000011 \;\; 1011100100010000000000000000000000000000000000000000000000$$

El bit de más a la izquierda es $\mathbf{s} = 0$ y nos indica un número positivo. Los siguientes 11 números (bits) 10000000011 son la característica ($\mathbf{c}$) y representa al número decimal:

$$c = 1 * 2^{10} + 0 * 2^9 + ... + 0 * 2^2 + 1 * 2^1 + 1 * 2^0 = 1024 + 2 + 1 = 1027$$

La parte exponencial del número es $2^{1027-1023} = 2^4$. Finalmente, los siguientes 52 bits representan la mantisa, la cual se representa como:

$$f = 1 * (\frac{1}{2})^1 + 1 * (\frac{1}{2})^3 + 1 * (\frac{1}{2})^4 + 1 * (\frac{1}{2})^5 + 1 * (\frac{1}{2})^8 + 1 * (\frac{1}{2})^{12}$$

Por lo tanto, el número de máquina (número punto flotante) representa el siguiente número decimal:

$$\begin{aligned}(-1)^s 2^{c-1023}(1+f) = & \\ &= (-1)^0 * 2^{1027-1023}(1 + \frac{1}{2} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{256} + \frac{1}{4096}) \\ &= 27{,}56640625\end{aligned}$$

Otra caracteristica importante en la notación científica es la **normalización**. Para normalizar, todo número real no nulo ($\neq 0$) se puede representar en forma única como:

$$\pm 0.d_1 d_2, ..., d_k \times 10^n, \;\; 1 \leq d_1 \leq 9 \;\; \text{y} \;\; 0 \leq d_i \leq 9$$

para cada $i = 1, ..., k$. Este tipo de números se les conoce como números de máquina decimales de digíto k

Ejemplos en R

Números binarios

Los dos tipos de almacenamiento de datos más comunes son los datos enteros y los datos numéricos. La mayoría de los números en R se almacenan como números de punto flotante.

Consideraciones: - R usa el formato de entero nativo de 32 bits. - El formato entero de 32 bits utiliza 31 bits para almacenar el número y un bit para almacenar el signo del número. - Como veremos, el formato de enteros no es el predeterminado en R. - El tipo de datos enteros no puede almacenar ninguna parte fraccionaria del número. - El número positivo más grande que R puede contener como un entero es 2^31 - 1 o 21474836477. A medida que incrementamos el número en uno, rápidamente llegamos a un número demasiado grande para el tipo de dato:

```
as.integer(2^31 - 2)
```

```
## [1] 2147483646
```

```
as.integer(2^31 - 1)
```

```
## [1] 2147483647
```

```
as.integer(2^31)
```

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

Podemos asegurarnos de tene run número entero agregando una "L" al número como:

```
-2147483646L
```

```
## [1] -2147483646
```

La letra "L" tiene como significado "*long*", el nombre de la declaración para un entero de 32 bits. Al eliminar la "L" final, las etapas intermedias se convierten en numéricas.

Aquí también se ignora el hecho de que el tipo de datos entero no puede almacenar números menores que 1. Por ejemplo, si ingresamos 1/2, se redondearía a cero.

```
as.integer(0.5)
```

```
## [1] 0
```

```
as.integer(1.7)
```

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

Números de punto flotante

Los números de punto flotante proporcionan una solución a las limitaciones de los números enteros binarios. Los números de punto flotante son capaces de almacenar valores no enteros tales como 3.51146985475, 1.9584462577 o 0.13. Los números de punto flotante también pueden almacenar números mucho más grande, como por ejemplo:

```
2^25
```

```
## [1] 33554432
```

```
2^35
```

```
## [1] 34359738368
```

Los números de punto flotante pueden incluir también números negativos de gran magnitud. Si no asignamos a un número como "entero" entonces `R` por defecto almacena los datos numéricos como números de punto flotante. Hay varios estándares para el punto flotante, pero nos centraremos en el número de punto flotante de doble precisión (*double*). Se llama precisión doble porque tiene aproximadamente el doble de espacio de almacenamiento disponible que el formato de número de punto flotante estándar (proveniente del lenguaje de programación `C`).

Dentro de `R` un número de punto flotante 1000 es almacenado como:

$$1000 = 0\text{b}1111101000 \times 2^9,$$

donde 2 nunca es almacenado y 0b1111101000 muestra la representación binaria de un número.

Otro ejemplo, considere el número 0.75, que puede ser representado como:

$$0{,}75 = \frac{3}{4} = 0\text{b}0001 \times 2^{-1},$$

Debido a que el número es una parte fraccionaria en base 2, cualquier número puede representarse como la suma de fracciones donde el denominador es una potencia de 2. La fracción 3/4 puede representarse por 1 / 2 + 1 / 4. Como decimales binarios, 1 / 2 = 0b0.1 y 1/4 = 0b0.01. Esto hace que 3/4 = 0b0.11.

El almacenamiento de números *double* tiene ventajas para un procesador numérico. Para sumar o restar, ambos números deben estar en un exponente común. Uno se puede convertir en el otro cambiando la mantisa. Después de esta conversión, los números se suman o restan, luego el resultado se almacena como un nuevo número de punto flotante.

Otro ejemplo, sumar 200 + 2000:

$$
\begin{aligned}
200 + 2000 &= 2 \times 10^2 + 2 \times 10^3 \\
&= 2 \times 10^2 + 20 \times 10^2 \\
&= 22 \times 10^2 \\
&= 2{,}2 \times 10^3
\end{aligned}
$$

Además de representar números, los números de puntos flotantes permiten almacenar valores especiales, y uno de los más importantes es `NaN`. El termino "NaN" significa "no es un número". Por ejemplo:

```
0 / 0
```

```
## [1] NaN
```

Para tener en consideración: `NaN` + x = `NaN`, para cualquier valor de x. `NaN`x, `NaN`*x y `NaN` = x, son todos iguales a `NaN` para cualquier valor de x (incluso si x es en sí mismo `NaN`). La función `is.nan()` permite verificar si un valor es `NaN`;

```
NaN == NaN
```

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

Otros ejemplo:

```
sqrt(-1)
```

```
## [1] NaN
```

```
Inf - Inf
```

```
## [1] NaN
```

Pero, si hacemos $\infty + \infty$

```
Inf + Inf
```

```
## [1] Inf
```

No podemos confundir `NaN` con `Na`. El término `Na` nos indica que existe ningún resultado disponible o que no existe un resultado en específico. Sin embargo, podemos utilizarlo, como por ejemplo, para llenar un vector:

```r
c(1, 2, 3, 4, 5, 6, NA)
```

```
## [1]  1  2  3  4  5  6 NA
```

o un valor dentro de un vector en una matriz:

```r
matrix(c(2, 3, NA, 5, 6, NA, NA, 8, 9), 3)
```

```
##      [,1] [,2] [,3]
## [1,]    2    5   NA
## [2,]    3    6    8
## [3,]   NA   NA    9
```

Los `NaN` difieren del "infitnito" (∞) y el cual tiene su propia representación. Dentro de `R` podemos encontrar dos formas de infinito, el positivo `Inf` y el negativo `-Inf`. El infinito puede ser obtenido a través de distintas operaciones o cuando se excede la capacidad de almacenamiento.

Para considerar, sobre los datos de tipo *double* es que la división entre valores numéricos 0 nos da un valor de `Inf`. Esto no se produco en otros marcos algebráicos, ya que el resultado es un número indefinido, por lo que tendría un mayor sentido a esta operación (y no otorgar `Inf`).

```r
1 / 0
```

```
## [1] Inf
```

o

```r
Inf / Inf
```

```
## [1] NaN
```

El `-Inf` se comporta de la misma manera que `Inf` pero con un comportamiento opuesto. Sin embargo, `Inf` también tiene la propiedad de ser igual a sí mismo pero no igual a `-Inf`. `Inf` es mayor que todos los demás números y `-Inf` es menor que todos los números:

```r
Inf == Inf
```

```
## [1] TRUE
```

```r
Inf == -Inf
```

```
## [1] FALSE
```

```r
Inf > 100
```

```
## [1] TRUE
```

```
Inf < 100
```

```
## [1] FALSE
```

```
-Inf > 100
```

```
## [1] FALSE
```

El infinito tambien sirve para mostrar cuando se excede el máximo mayor número que puede ser almacenado. Por ejemplo:

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

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

```
1 +     .Machine$double.eps != 1
```

```
## [1] TRUE
```

```
1 + .5* .Machine$double.eps == 1
```

```
## [1] TRUE
```

¿Por que no entrega como resultado `.Machine$double.xmax` infinito (`Inf`)?

Para finalizar, una de las caracteristicas del tipo de datos de doble precisión (*double*) es la existencia del 0 con un signo. El 0 negativo, -0, actúa como 0, pero el signo negativo tiende a propagarse como se espera:

```
1 / -0
```

```
## [1] -Inf
```

Si un resultado es -0 entonces se mostrará como 0, mientras el negativo todavía se lleva en el número y se usa en los cálculos. Esto puede llevar a un resultados algo contradictorios:

```
-0
```

```
## [1] 0
```

```
1 / -Inf
```

```
## [1] 0
```

```
1 / sqrt(-0)
```

```
## [1] -Inf
```

de lo contrario, -0 actua como 0 y -0 = 0, pero si consideramos $x - x$ siempre es 0 y no -0, por lo que es algo confuso.

Referencias

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

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