NOTAS DE ANÁLISIS NUMÉRCO

ISRAEL GARCÍA
UNIVERSIDAD AUTÓNOMA DE YUCATÁN

1. Preliminares

1.1. Distintos tipos de bases. Considera el numero $x = 123456$. Aunque es claro que significa x, formalmente debemos entender que

$$x = 1 \cdot 10^5 + 2 \cdot 10^4 + 3 \cdot 10^3 + 4 \cdot 10^2 + 5 \cdot 10 + 6,$$

pues de esta forma, los resultados previos de matematicas como sucesiones y series se pueden utilizar para analizar las propiedades de los numeros. Sin embargo, en analisis numerico no es conveniente utilizar potencias de 10 para representar los numeros, pues la computadora utiliza cables y electricidad para funcionar, de modo que es facil representar 0 y 1 como la presencia/ausencia de electricidad.

Teorema 1. *Para todo numero x existe una sucesion*

$$a_0, a_1, a_n \dots$$

tal que

$$x = \sum_{i=-\infty}^{n} a_i \cdot b^i,$$

donde

$$b > 1$$

es un numeronatural, llamado la base.

La sucesion

$$a_0, a_1, \dots$$

del teorema anterior suele expresarse asi:

$$x = (a_n, \dots, a_1, a_0, \dots)_b$$

En particular si $b = 2$, la base se llama binaria, y si la base es decimal, entonces por definicion

$$123456 = (1, 2, 3, 4, 5, 6)_{10}$$

Ejercicio 1. Investiga o demuestra que si en algun momento la sucesion

$$\{a_k\}$$

se convierte en una sucesion periodica, el numero x es racional. El reciproco tambien es cierto, pero es mas dificil de probar.

1.2. Numeros binarios. En general, la computadora guarda todos los datos en *binario*, de modo que existen varios estandares para representar los diferentes tipos de datos: *caracteres, enteros, flotantes, etc.*, cada uno tiene un tamaño predefinido. Por otro lado, representar números binarios, o pensar en el valor de un número en potencias de dos puede resultar laborioso, pues por ejemplo, el número 95578, tiene la representación binaria 10111010101011010, mucho más larga que en decimal, de modo que es más conveniente utilizar alguna otra base para expresar el número de manera humanamente entendible. Por razones históricas, se acostumbra representar los datos de la computadora en *formato hexadecimal*.

Definición 1. La notación $0b\, a_1 \dots a_n$ representa un número en formato binario, y la notación $0x\, a_1 \dots a_k$ lo representa en hexadecimal, donde para el caso binario los dígitos $a_i \in \{0, 1\}$, pero para evitar confusión, en el caso hexadecimal los dígitos $a_i \in \{0, \dots, 9, a, b, c, d, e, f\}$.

Así, en nuestro ejemplo

$$95578 = 0b\, 10111010101011010 = 0x\, 1755a,$$

o en *python*:

```
>> x = 95578
>> bin(x)
10111010101011010
>> hex(x)
1755a
```

1.3. Numeros de punto flotante.

Definición 2. Un número x se expresa en notación científica si puede escribirse de la forma

$$x = \pm r \times 10^n,$$

donde $1/10 \leq r < 1$. El número r se llama la *mantissa normalizada*

Esta definición se extiende de manera natural a cualquier otra base para representar al número, en particular, en binario

$$x = \pm r \times 2^n,$$

donde $1/2 \leq r < 1$. Como la computadora tiene una capacidad de almacenamiento finito, en realidad, solo una pequeña parte de los números reales puede representarse dentro de la computadora con números de punto flotante.

Ejercicio 2.
- Cuántos números de punto flotante pueden expresarse con una mantissa de tres *bits* y un exponente de tres bits:

  $$x = \pm(0.b_1 b_2 b_3) \times 2^{\pm k},$$

  $(k, b_i \in \{0, 1\})$.
- Con hipotesis similares, cuántos números pueden escribirse en base tres?.
- Dibuja los puntos diferentes que obtuviste, de preferencia, utiliza la computadora para hacer esto.

Este ejercicio podria ser más fácil de resolver con un programa.

Definición 3. Si en el transcurso de un cálculo, se produce un número flotante $\pm r \times 2^m$, donde m está fuera del rango permitido, se dice que se tiene un *overflow* o *underflow*. En general, ambos casos se llaman un *desbordamiento*.

Definición 4. Un número flotante de punto fijo es un número real representable en la computadora utilizando una palabra de 32-bits

Cómo se representa un número flotante depende de la implementación de la computadora, pero en general se acostumbra seguir el estandard de la *IEEE*. Dado que un número flotante es de la forma

$$\pm q \times 2^m,$$

En general, la forma de la representación es

$$(-1)^s \times 2^{c-127} \times (1.f)_2,$$

es decir, que el bit más a la izquierda es utiliza para determinar el signo de la mantissa, $s = 0$ corresponde a $+$ y $s = 1$ corresponde a $-$. Los siguientes ocho bits corresponden al entero no negativo c. Finalmente, los últimos 23 bits corresponden a f en la mantissa *normalizada*, pues como el primer dígito siempre es 1, no es necesario guardarlo.

Observación 1. Hay una excepcion a esta regla, pues para el número 0 son ambiguos tanto el signo, como el exponente y la mantissa no puede tener un 1, sin embargo es bastante obvio cómo representarlo. (*cómo?*)

Definición 5. El *epsilon de la máquina* es el número flotante más pequeño tal que $1 + \epsilon \neq 1$.

Esta definición hace sentido, porque los números flotantes son finitos. Para un número de precisión simple (32 bits) se tiene $\epsilon = 2^{-23}$. (*porque?*).

Observación 2. Como $2^{-23} \approx 1.2 \times 10^{-7}$, en un cálculo simple, con números flotantes de precisión simple, aproximadamente se obtienen seis cifras decimales exactas.

1.3.1. Números flotantes de precisión doble. Cuando se necesita más precisión que la que se puede obtener con los números de precisión simple, se utiliza los de precisión doble, estos son números flotantes con mantissa de 52 bits.

Ejercicio 3. Cuál es la precisión de un cálculo simple con flotantes dobles?

Observación 3. En *python* los números tienen precisión arbitraria lo cual es muy costoso. En bibliotecas numéricas como *numpy* los números flotantes tienen precisión doble por default.

Ejercicio 4.
- Determina la representación como número flotante simple del número -23.1432.
- Determina qué número real corresponde al flotante $[2340ED01]_{16}$?

Observación 4. Es posible verificar la respuesta del segundo ejercicio en *python* de la siguiente manera:

```
1  import struct
2
3  x = '\x23\x40\xED\x01'
4  struct.unpack('!f', x)
```

Referencias

[1] E. Cheney and D. Kincaid. *Numerical Mathematics and Computing*. International student edition. Cengage Learning, 2007.