Física Computacional 2 - LFIS-126

Profesor: Julio C. Marín

Departamento de Meteorología, Universidad de Valparaíso

Segundo Semestre

Temas de la clase

- Precisión y rapidez
- Rango de variables
- Errores numéricos

Precisión y rapidez

- Computadoras tienen limitaciones.
- No pueden guardar números reales con un número infinito de lugares decimales.
- Hay un límite en el número más grande y el más pequeño que pueden guardar.
- Esto no es importante en mayoría de aplicaciones.
- Pero hay situaciones donde estas limitaciones nos afectan notablemente.
- Necesitamos entender limitaciones así como métodos para atenuar estos efectos.

Rango de variables (flotantes)

- Hay un rango (min - máx) de valores que Python puede aceptar como enteros y números flotantes.

código Python

-> import sys as sys

Rango de variables (flotantes)

- Hay un rango (min - máx) de valores que Python puede aceptar como enteros y números flotantes.

código Python

-> import sys as sys

Números flotantes

-> sys.float_info

-> sys.float_info(max=1.7976931348623157e+308, max_exp=1024, max_10_exp=308, min=2.2250738585072014e-308, min_exp=-1021, min_10_exp=-307, dig=15, mant_dig=53, epsilon=2.220446049250313e-16, radix=2, rounds=1)

Rango de variables (flotantes)

- sys.float_info.max (1.7976931348623157e+308): No. flotante + máximo que puede ser representado
- radix: Base de la representación exponencial
- min (2.2250738585072014e-308): No. flotante + min. que puede ser representado.
- max_exp(e): Máx. No. entero tal que radix**(e - 1) puede representarse como No. flotante
- Max valores son: ≈10.0**308 y 2.0**1023

Rango de variables (flotantes)

Ejercicios:

- Cuál es el No. máximo al que se puede elevar 5.523 hasta que de error?
- Cuál es el No. máximo al que se puede elevar 10.245 hasta que de error?

Rango de variables (flotantes)

Ejercicios:

- Cuál es el No. máximo al que se puede elevar 5.523 hasta que de error?
- Cuál es el No. máximo al que se puede elevar 10.245 hasta que de error?

```
for ii in range(300, 800):
    print("Exponente ", ii)
    print(5.523**ii)
```

Variable excedida (overflowed) al correr programa

Si se excede valor máx. permitido:

Si valor de variable excede No. flotante más grande que puede almacenar la PC, => error: Variable se ha excedido (overflowed)

Valor más pequeño que puede ser representado como flotante

- Potencia a la -****
- Se asigna 0 si valor es más pequeño que este número

Rango de variables (enteros)

- No hay límites en Python para representar enteros.
- Depende de cuanta memoria tengas disponible.
- Sin embargo, cálculos con enteros muy grandes puede tomar mucho tiempo.
- Probemos los siguientes ejemplos:
  - 2**10000
  - 2**100000
  - 2**1000000

Ejercicio

- Escribir programa para calcular y mostrar el factorial de un número entero. Qué pasa al correr el programa?

Ejercicio

- Escribir programa para calcular y mostrar el factorial de un número entero. Qué pasa al correr el programa?

```
def factorial(x):
....: f = 1
....: for k in range(1,x+1):
....:         f *= k
....: return f
for ii in range(100, 1500):
....: print('El factorial para ', ii, ' = ', factorial(ii))
```

Ejercicio

- Modifique su programa para que calcule el factorial de un número flotante. Cuál es el mayor número al que se puede calcular el factorial sin indefinirse?

Ejercicio

- Modifique su programa para que calcule el factorial de un número flotante. Cuál es el mayor número al que se puede calcular el factorial sin indefinirse?

```
def factorial(x):
....: f = 1.0
....: for k in range(1,x+1):
....:         f *= k
....: return f
for ii in range(1, 180):
....: print('El factorial para ', ii, ' = ', factorial(ii))
```

Error numérico

- No. flotantes se representan en una PC con una precisión limitada.
- No ocurre con los enteros.
- En Python se usa un nivel de precisión de 16 dígitos significativos.

Valor real de π: 3.14159265358979323846 26 ...
Valor en Python: 3.141592653589793
Diferencia: 0.0000000000000002384626...

Error numérico

- No. flotantes se representan en una PC con una precisión limitada.
- No ocurre con los enteros.
- En Python se usa un nivel de precisión de 16 dígitos significativos.

Valor real de π: 3.14159265358979323846 26 ...
Valor en Python: 3.141592653589793
Diferencia: 0.00000000000000002384626...

Error de redondeo

Error en la representación del número por la PC

Error numérico

- Si sumamos 1.1 + 2.2 debe dar 3.3

  -> 1.1 + 2.2 = 3.3000000000000003

- Por qué ocurre esto?
- No. flotantes se representan en hardware the PC como fracciones con base 2.
- Como se representa 0.125 en notación fraccional con base 10: $1/10 + 2/100 + 5/1000$
- Como se representa 0.125 en notación fraccional con base 2: $0/2 + 0/4 + 1/8$

- Mayoría de fracciones decimales no pueden ser representadas exactamente como fracciones binarias.
- No. flotantes decimales son aproximados por No. flotantes binarios en PC.

Error numérico

- Nunca debemos usar una declaración IF para probar igualdad de dos No. flotantes
- El siguiente código puede dar error:

```
-> if x == 3.3:
        print(x)
```

- Se debe reemplazar por algo como:

```
-> eps = 1e-12
-> if (abs(x - 3.3) < eps):
        print(x)
```

Error numérico

Error de redondeo (ϵ) es la cantidad que hay que sumar al valor calculado por la PC para obtener el valor real.

-> from math import sqrt

-> x = sqrt(2)

Resultado no será $x = \sqrt{2}$ sino $x + \epsilon = \sqrt{2} \Rightarrow x = \sqrt{2} - \epsilon$

Error numérico

Error de redondeo (ϵ) es la cantidad que hay que sumar al valor calculado por la PC para obtener el valor real.

-> from math import sqrt

-> x = sqrt(2)

Resultado no será $x = \sqrt{2}$ sino $x + \epsilon = \sqrt{2} \Rightarrow x = \sqrt{2} - \epsilon$

Similar definición de error que cuando se dice la edad del universo es de 13.75 ± 0.11 billones de años.

Error numérico

- Gran problema puede ocurrir al sustraer números similares

Ejemplo:

$$x = 1000000000000000$$

$$y = 1000000000000001.2345678901234$$

- Al calcular $y - x$ la PC solo representa los números con 16 cifras significativas

$$x = 1000000000000000$$

$$y = 1000000000000001.2$$

Error numérico

$$x = 1000000000000000$$

$$y = 1000000000000001.2$$

- x es representado exactament pero y fue truncado
- $y - x = 1.2$ cuando debía ser 1.2345678901234

En vez de exactitud de 16 cifras significativas, tenemos exactitud de 2 y el error fraccional aumenta.

Error numérico

$$x = 1000000000000000$$

$$y = 1000000000000001.2$$

- x es representado exactament pero y fue truncado
- $y - x = 1.2$ cuando debía ser 1.2345678901234

En vez de exactitud de 16 cifras significativas, tenemos exactitud de 2 y el error fraccional aumenta.

Si diferencia entre 2 No. es muy pequeña, comparable con la exactitud de la PC, el error fraccional puede ser grande.

Error numérico

Ejemplo en python

$$x = 1$$

$$y = 1 + 10^{-14}\sqrt{2}$$

$$10^{14}(y - x) = \sqrt{2}$$

-> from math import sqrt

-> $x = 1$; y=1.0+(1e-14)sqrt(2)

-> print((1e14)*(y-x))

-> print(sqrt(2))

Error numérico

Ejemplo en python

$$x = 1$$

$$y = 1 + 10^{-14}\sqrt{2}$$

$$10^{14}(y - x) = \sqrt{2}$$

-> from math import sqrt

-> $x = 1$; y=1.0+(1e-14)sqrt(2)

-> print((1e14)*(y-x))

-> print(sqrt(2))

1.42108547152

1.41421356237

Cálculo exacto solo hasta la primera cifra decimal

Error numérico

- Cálculos que conllevan substracción de No. casi iguales, producen grandes errores.
- Aparece con frecuencia en cálculos de física
- Es quizás la causa más común de errores numéricos significativos en cálculos

Velocidad de un programa

- PCs no son ∞ exactas!!
- PCs no son ∞ rápidas!!
- 1 millón de operaciones matemáticas toma menos de 1 seg.
- 1 billón de operaciones matemáticas toma min. u horas.
- 1 trillón de operaciones matemáticas toma una eternidad!!! :)

Velocidad de un programa

- PCs no son ∞ exactas!!
- PCs no son ∞ rápidas!!
- 1 millón de operaciones matemáticas toma menos de 1 seg.
- 1 billón de operaciones matemáticas toma min. u horas.
- 1 trillón de operaciones matemáticas toma una eternidad!!! :)

Regla general

- Puedo efectuar cálculos en una PC, que tomen un 1 billón de operaciones o menos.
- Depende si estoy haciendo sumas o multiplicaciones o calculando func. Bessel, multiplicación de matrices, etc.

Ejemplo: Oscilador armónico quántico a Temp. finita

- El oscilador armónico quántico simple tiene niveles de energía: $E_n = \hbar\omega(n + \frac{1}{2})$
- Boltzmann y Gibbs mostraron que a la temp. T, el oscilador tenía una energía promedio dada por:

$$\langle E \rangle = \frac{1}{Z} \sum_{n=0}^{\infty} E_n e^{-\beta E_n}$$

  donde $\beta = 1/(k_B T)$, k_B es la cte de Boltzmann y $Z = \sum_{n=0}^{\infty} e^{-\beta E_n}$.
- Calcular el valor de $\langle E \rangle$ cuando $k_B T = 100$
- Tomemos primeros 1000 términos de suma y supongamos unidades tales que $\hbar = \omega = 1$

Ejemplo: Oscilador armónico quántico a Temp. finita

```
-> from math import exp
-> terms = 1000; beta = 1/100; S = 0.0; Z = 0.0;
-> for n in range(terms):
...        En = n + 0.5
...        weight = exp(-beta*En)
...        S += weight*En
...        Z += weight
-> print(S/Z)
```

Ejemplo: Oscilador armónico quántico a Temp. finita

```
-> from math import exp
-> terms = 1000; beta = 1/100; S = 0.0; Z = 0.0;
-> for n in range(terms):
...        En = n + 0.5
...        weight = exp(-beta*En)
...        S += weight*En
...        Z += weight
-> print(S/Z)
```

- Ctes (β y No. de términos) se asignan a inicio de programa. Buen estilo de programación
- Se usa un solo ciclo for para hacer los cálculos. Ahorra tiempo de cómputo
- Término $\exp(-\beta E_n)$ se calcula 1 vez en cada ciclo. Ahorra tiempo. Más rapido +, -, * y / que calcular exp()

Ejemplo: Oscilador armónico quántico a Temp. finita

- Si corremos programa => 99.9554313409

Ejemplo: Oscilador armónico quántico a Temp. finita

- Si corremos programa => 99.9554313409
- Aumentemos el No. de Términos (terms) para mayor exactitud en estimación de $\langle E \rangle$
  - terms = 1 millón: 1.000.000 => 100.000833332
  - Cambia resultado y veloc. de cómputo no aumenta mucho!!!

Ejemplo: Oscilador armónico quántico a Temp. finita

- Si corremos programa => 99.9554313409
- Aumentemos el No. de Términos (terms) para mayor exactitud en estimación de $\langle E \rangle$
  - terms = 1 millón: 1.000.000 => 100.000833332
  - Cambia resultado y veloc. de cómputo no aumenta mucho!!!
- terms = 1billón: 1.000.000.000
  - Cálculo toma más de 20 min. pero resultado no cambia mucho

Debemos sopesar el balance entre tiempo de cómputo y exactitud del cálculo. Decidir cuando uno es más importante que el otro

Ejercicio 4.2. Ecs. cuadráticas

- Escriba programa que tome 3 números como entrada: a, b y c, e imprima las soluciones de ecuación cuadrática $ax^2 + bx + c = 0$ usando la fórmula estándar:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- Calcule las soluciones de $0.001x^2+1000\mathrm{x}+0.001=0$

Ejercicio 4.2 del libro

Ejercicio 4.2. Ecs. cuadráticas

- Escriba programa que tome 3 números como entrada: a, b y c, e imprima las soluciones de ecuación cuadrática $ax^2 + bx + c = 0$ usando la fórmula estándar:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- Calcule las soluciones de $0.001x^2$+1000x+0.001=0

- Multiplicando numerador y denominador por $-b \mp \sqrt{b^2 - 4ac}$ se llega a otra ec. para raíces de ec. cuadrática:

$$x = \frac{2c}{-b \mp \sqrt{b^2 - 4ac}}$$

- Repetir el ejercicio con esta nueva fórmula.

Tarea!!! Ejercicio 4.3 del libro

Ejercicio 4.3. Calcular derivadas