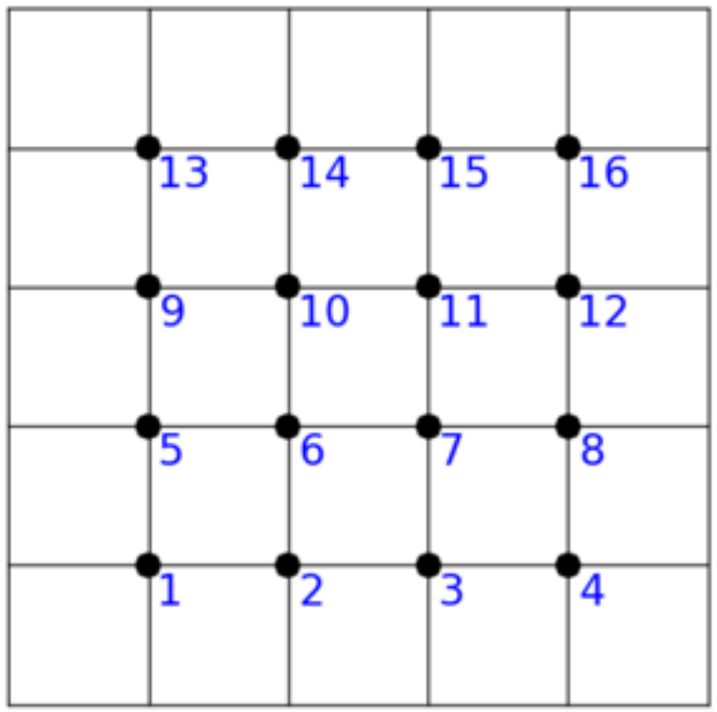
Memoria/ Números/ Python

Métodos Computacionales
Semana 2

Ejemplo: **Problema de Conducción Térmica (Estado Estacionario)**

- Vamos a discretizar un sistema 2D en donde hay conducción térmica.



- ¿Cuánto ocuparía el almacenamiento de una matriz de ~10000 x 10000?

Memoria

- Depende del número de bits/bytes
- 1 byte = 8 bits (*binary digit* 1/0)
- Dejamos 8 bytes para cada punto de la malla

8×10^8 bytes = 800 MB

¿Qué porcentaje de valores en este arreglo son != 0?

Memoria

¿Cuántos números distintos puede almacenar un byte?

- Desde 00000000 hasta 11111111

- Para cada número real usamos normalmente 4 bytes (32 bits) u 8 bytes (64 bits)

Enteros

- Representación de “complemento a dos”
- Usamos el primer bit: 0=+, 1=-
- La suma funciona directamente (¿por qué?)

Enteros

- En representación de 32 bits, tenemos valores entre -2^{31} hasta 2^{31}.

- En Python, el software automáticamente guarda enteros más grandes (¡pero la aritmética es más lenta!)

```
$ python

>>> 2**30

>>> 2**100
```

Notación de punto fijo

- 64 bits, repartidos entre *N* para la parte entera y *M* para la parte fraccionaria.
- Se usan 5 (entera) + 6 (fraccionaria).

¿Cómo representamos π en esta notación?

Notación de punto fijo

Problemas:

- Se desperdician bits (con ceros a la derecha)
- La precisión depende del tamaño del número
- Aritmética por software
- Rango limitado

Notación de punto flotante

- Idea: Usar notación científica

$$0.2345\text{e}{-18} = 0.2345 \times 10^{-18} = 0.000000000000000000002345$$

- Mantisa = 0.2345 Exponente: -18

- Ejemplo: Mantisa = 0.101101, Exponente = -11011

$$\begin{aligned} 0.101101 &= 1(2^{-1}) + 0(2^{-2}) + 1(2^{-3}) + 1(2^{-4}) + 0(2^{-5}) + 1(2^{-6}) \\ &= 0.703125 \ \text{(base 10)} \end{aligned}$$

$$-11011 = -1(2^4) + 1(2^3) + 0(2^2) + 1(2^1) + 1(2^0) = -27 \ \text{(base 10)}$$

- **El número es**

$$0.703125 \times 2^{-27} \approx 5.2386894822120667 \times 10^{-9}$$

Reales en Python

- Estándar IEEE
- 53 bits para la mantisa
- 11 bits para el exponente

¿Cuántos dígitos de precisión?

Reales en Python

- ¡Cuidado con las bases!

```
>>> from numpy import pi

>>> pi

>>> 1000 * pi

>>> pi/1000
```

Python - Lo Bueno

- Actualizado continuamente
- Módulos para temas generales (Matplotlib, NumPy, SciPy)
- Módulos para temas especializados (AstroPy, SunPy,…)
- Inicialmente intuitivo
- Hay muchos *shells* (intérpretes) interactivos
- Fácil de combinar con el *shell* de Unix y con otros lenguajes
- Open source
- Usualmente su uso es de alto nivel

Python - Lo Malo

- Descentralizado
- Documentación “en la nube”
- Puede ser lento para manejar *arrays* (arreglos)
- Es interpretado, no compilado

Lenguaje orientado a objetos

```
>>> x = 3.4

>>> print id(x), type(x)

>>> x = 5

>>> x = 5.

>>> x = [4,5,6]

>>> x = [7,8,9]

>>> x = “La suya”
```

Lenguaje orientado a objetos

```
>>> x = [7,8,9]

>>> x.append(10)

>>> x

>>> print id(x), type(x)
```

Las listas son objetos mutables

Lenguaje orientado a objetos

```
>>> x = [7,8,9]

>>> print id(x)

>>> y = x

>>> print id(y)
```

```
>>> y.append(27)

>>> x
```

`x, y` son punteros al mismo objeto “lista”

Lenguaje orientado a objetos

```
>>> x = [7,8,9]

>>> print id(x)

>>> y = list(x)
```

```
>>> y

>>> print id(y)

>>> y.append(27)

>>> x
```

`x, y` son punteros al mismo objeto “lista”

Integers/floats no son mutables

```
>>> x = 3.4

>>> print id(x),x

>>> y = x

>>> print id(y),y

>>> y = y+1

>>> print id(x),id(y),x,y
```

Python protege los objetos no mutables automáticamente

Ejemplo: Raíces Cuadradas

- Solución aritmética finita no es posible
- En Google: sqrt 2
- En Python:

```
>>> sqrt(2.)

>>> from numpy import sqrt

>>> sqrt(2.)
```

El método de Newton

$$s^2 - x = 0$$

```
s = 1.     # conjetura inicial
for k in range(kmax):
    s = 0.5 * (s + x/s)
```

- Es necesaria la indentación
- range(N) produce la lista [0, 1, 2,…, N-1]

¿Cómo funciona este algoritmo?

El método de Newton

$$s^2 - x = 0$$

- Si s es un cero de $f(s)$, el punto de corte de la recta tangente a $f(x)$ en s es cero.

- A medida que me acerco a la solución, la aproximación lineal funciona mejor.

- Converge cuadráticamente (los errores se elevan al cuadrado en cada iteración).

El método de Newton

$$s^2 - x = 0$$

- Extraemos la siguiente iteración

$$s^{[k+1]} = s^{[k]} - \frac{f\left(s^{[k]}\right)}{f'\left(s^{[k]}\right)}$$

- Simplificando para nuestro problema:

```
s = 0.5 * (s + x/s)
```

Ejemplo: **Problema de Conducción Térmica (Estado Estacionario)**

$$u_{i,j} = \frac{1}{4}(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1})$$

Sistema de N^2 ecuaciones con N^2 incógnitas.

Hay que solucionar la ecuación matricial **Au**=**B**

La matriz **A** tiene dimensiones N^2 x N^2

Es una matriz dispersa, o rala