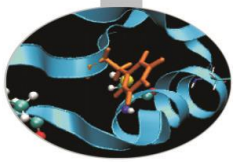
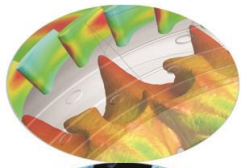
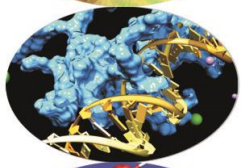
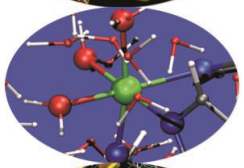
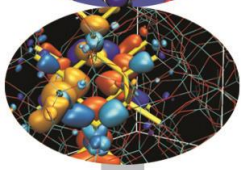
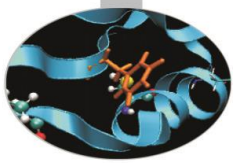


Esercitazione

Numpy

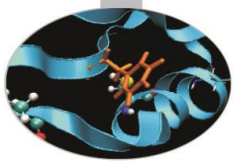


Esercitazione

- Esercizio 1 (Numpy sintassi base)

Lo scopo di questo esercizio è di familiarizzare con la sintassi degli array. Completare l'esercizio come richiesto nel testo numpy_base.py

(Soluzione numpy_base.py)



Esercitazione

Esercizio 2 (dtype):

Creare un nuovo dtype ‘atom’ per rappresentare un atomo tridimensionale, contenente:

-stringa (‘S3’) per rappresentare il simbolo

-Intero ('PA') per il peso atomico

-3 float64 per rappresentare le coordinate cartesiane

Creare un array con dtype=atom per rappresentare la molecola d’acqua H2O (O coordinate 0,0,0, H coordinate 0,0,1.89, H coordinate 1.861,0,-0.328)

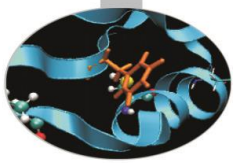
Caricare il file mol.xyz contenente un array di dati di tipo atom.

Stampare il contenuto dell'array così formato.

Estrapolare l'array dei pesi atomici e calcolare il Peso Molecolare $MW = \sum_i PA_i$

Calcolare inoltre il contributo di ogni singolo atomo nel calcolo del peso molecolare

Creare uno nuovo array con le sole coordinate degli atomi, stampare la shape e il valor medio $(\bar{x}, \bar{y}, \bar{z})$



Esercitazione

Calcolare il centro di massa molecolare

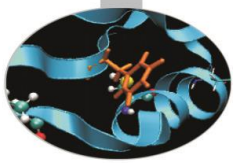
$$xcm = \frac{1}{MW}\sum_i x_i m_i \qquad ycm = \frac{1}{MW}\sum_i y_i m_i \qquad zcm = \frac{1}{MW}\sum_i z_i m_i$$

Calcolare il tensore del momento di inerzia:

$$I = \begin{bmatrix} I_{11} & I_{12} & I_{13} \\ I_{21} & I_{22} & I_{23} \\ I_{31} & I_{32} & I_{33} \end{bmatrix}$$

$$I_{11} = I_{xx} = \sum_i m_i (y_i^2 + z_i^2)$$
$$I_{22} = I_{yy} = \sum_i m_i (x_i^2 + z_i^2)$$
$$I_{22} = I_{zz} = \sum_i m_i (x_i^2 + y_i^2)$$
$$I_{12} = I_{xy} = -\sum_i m_i x_i y_i$$
$$I_{13} = I_{xz} = -\sum_i m_i x_i z_i$$
$$I_{23} = I_{yz} = -\sum_i m_i y_i z_i$$

(Soluzione dtype.py)



Esercitazione

Esercizio 3 (sistema.py)

Risolvere il seguente sistema di equazioni Ax=y usando Numpy, lavorare con il modulo `numpy.linalg`. Calcolare inoltre autovalori e autovettori della matrice A

$$2 \cdot x_1 + x_2 + x_3 = 9$$

$$x_1 + 2 \cdot x_2 + x_4 = 8$$

$$x_1 + 2 \cdot x_2 + 3 \cdot x_3 + 2 \cdot x_4 = 7$$

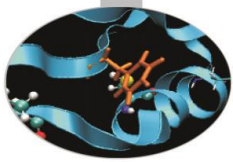
$$2 \cdot x_2 + x_3 + 2 \cdot x_4 = 6$$

Esercizio 4 (ruota.py)

Scrivere una funzione per ruotare un vettore 2D di un angolo theta

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Esercitazione



Esercizio 5 (vectorization)

Calcolare l'integrale Monte-Carlo di una funzione data (f(x)=1+2x) dove i punti xi sono numeri random uniformemente distribuiti nell'intervallo a, b. Elaborare un programma python che implementi tale integrazione con un loop esplicito e tramite vettorizzazione via numpy. Infine, testare il cpu time delle due porzioni del programma tramite la funzione time.clock()

$$\int_a^b f(x) \approx \frac{b-a}{n} \sum_{i=1}^{n} f(x_i)$$

(Solution: intMC.py)

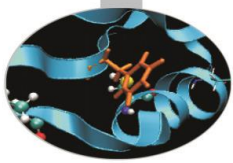
Esercizio 6 (vectorization)

Creare due array x e h(x), dove x è un intervallo equispaziato tra [-4,4] e `h(x)` è definita da:

$$h(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$

Creare due array tramite la funzione `zeros` e riempirli con un loop for.

Successivamente vettorizzare il codice con la funzione `linspace`. Verificare che i risultati siano identici. Valutare la cpu time delle due versioni con la funzione `time.clock`, all'aumentare del numero di intervalli. (Solution: vectorize.py)



Esercitazione

Esercizio 7 (vectorization)

Implementare due versioni della seguente regola di integrazione. Tramite un ciclo for e vettorizzando tramite slicing. Valutare il tempo di calcolo delle due versioni. Usare la funzione f(x)=1+2x tra [1,10] , come funzione di prova. (Solution: integrate_rule.py)

$$\int_a^b f(x)dx \approx \frac{h}{2}f(a) + \frac{h}{2}f(b) + h\sum_{i=1}^{n-1} f(a+ih), \quad h = \frac{b-a}{n}.$$

Esercizio 8 (vectorization)

La formula ricorsiva:

$$u_{i,j}^{\ell+1} = \beta(u_{i-1,j}^{\ell} + u_{i+1,j}^{\ell} + u_{i,j-1}^{\ell} + u_{i,j+1}^{\ell}) + (1-4\beta)u_{i,j}^{\ell}$$

È una generalizzazione al caso bidimensionale dello schema numerico di diffusione del calore visto a lezione. Considerare un array bidimensionale (100,100) per rappresentare u e applicare lo schema ricorsivo. Usare un doppio ciclo for e successivamente vettorizzare l'espressione tramite slicing di array.

Calcolare i tempi di calcolo delle due versioni. (Solution: slicing_2D.py)