ALMA MATER STUDIORUM
UNIVERSITA' DI BOLOGNA

SCUOLA DI INGEGNERIA
Sede di Forlì

Corso di Laurea in
INGEGNERIA MECCANICA
Classe L-9

ELABORATO FINALE DI LAUREA
In Macchine

# SVILUPPO DI UN FRAMEWORK DI ALGORITMI PER L'ANALISI DEL SEGNALE DI PRESSIONE NEL CILINDRO

CANDIDATO

Pugliese Giovanni Raffaele

RELATORE

Professor Corti Enrico

Anno Accademico 2019/2020

# Sommario

# 1

# INTRODUZIONE

## 1.1. Motori a Combustione Interna e loro classificazione

I motori a combustione interna sono delle macchine motrici termiche. La definizione motrice deriva dal fatto che essi hanno come scopo quello di convertire l'energia liberata dalla combustione di una carica, composta da combustibile e aria, in lavoro. Per quanto riguarda invece la definizione di interna, essa è legata al fatto che la combustione è sviluppata tutta all'interno della macchina stessa e la sorgente di calore è contenuta all'interno del fluido processato. Per andare a classificare i vari MCI usiamo dei parametri che ci aiutano a classificarli in base al loro comportamento:

- *modo di avviare la combustione del fluido:* li possiamo suddividere in motori ad accensione comandata in cui l'accensione avviene in seguito allo scoccare di una scintilla che avvia la combustione del fluido elaborato e i motori ad accensione spontanea o accensione per compressione, in questo caso il fluido inizia la combustione grazie ad un repentino aumento di pressione nel cilindro che fa iniziare la combustione spontanea del fluido che precedentemente era stato finemente polverizzato in camera.
- *durata del ciclo:* questo parametro ci descrive in quante corse del pistone si completa un ciclo e ci permette di suddividere i motori in: due tempi, ciclo completato in due corse e quattro tempi, ciclo completato in quattro corse.
- *natura del combustibile:* ci permettono di suddividerli in base alla natura chimica del combustibile.
- *alimentazione dell'aria:* motori aspirati, in cui la pressione di aspirazione dell'aria è uguale alla pressione atmosferica, motori sovralimentati, in cui la pressione di aspirazione dell'aria è uguale alla pressione di sovralimentazione.
- *alimentazione del combustibile:* li suddividiamo in base al modo in cui la miscela A/F (Air/Fuel) viene portata nel cilindro e li dividiamo in: motori a carburatore, motori ad iniezione interna o GDI (Gasoline Direct Injection) e motori ad iniezione indiretta o PFI (Port Fuel Injection).

## 1.2. Parametri caratteristici

I parametri fondamentali che mi consentono di caratterizzare un motore a combustione interna azionato da un manovellismo di spinta centrato sono:

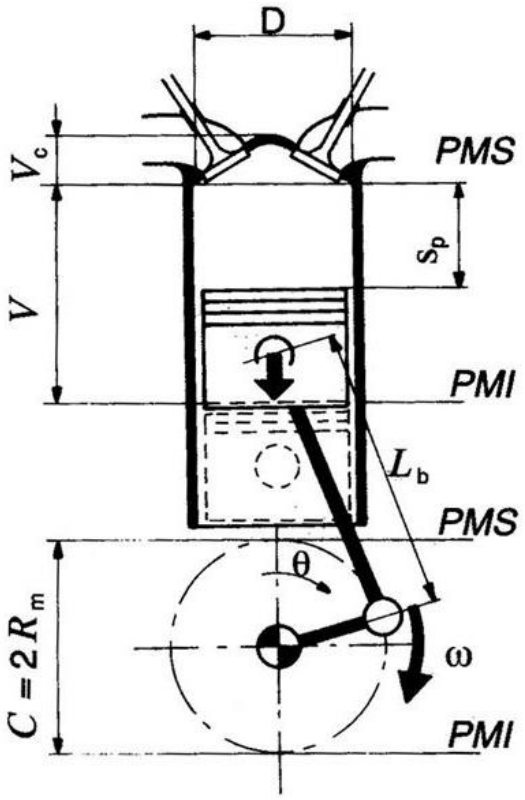



*Figura 1: Cilindro, Pistone e manovellismo*

- Alesaggio D: esso corrisponde al diametro interno del cilindro in cui scorre il pistone.

- Corsa C: è la distanza che spazza il pistone durante un giro motore e va dal PMS (Punto Morto Superiore) al PMI (Punto Morto Inferiore), essa si calcola:

$$C=2R_m \qquad (1.1)$$

- Sezione trasversale del cilindro $A_c$: area della sezione del cilindro ed è uguale a:

$$A_c = \pi D^2/4 \qquad (1.2)$$

- Cilindrata V: è la variazione di volume prodotta dal pistone durante la sua corsa:

$$V = A_c C = \pi D^2 C/4 \qquad (1.3)$$

- Rapporto volumetrico di compressione $r_c$: è il rapporto tra il massimo valore del volume e il minimo valore. Il massimo valore è raggiunto quando il pistone si trova al PMI e corrisponde a tutto il volume del cilindro a cui si aggiunge il volume della camera di combustione ($V_c$), mentre il minimo valore è raggiunto al PMS e corrisponde semplicemente al volume della camera di combustione. La formula è:

$$r_c = (V+V_c)/V_c \qquad (1.4)$$

- Volume istantaneo del cilindro **V**: rappresenta il volume in funzione dell'angolo di manovella e si calcola con la seguente formula:

$$\boldsymbol{V}=V\{1/(r-1) + 1/2[1+1/\gamma - \cos\theta - 1/\gamma\sqrt{(1-\gamma^2 sin^2\theta}]\} \qquad (1.5)$$

*con* $\gamma = r/l$

Altri parametri caratteristici che mi consentono di classificare i motori sono la potenza effettiva $P_e$ e la coppia effettiva $M_e$. Queste due grandezze derivano da prove sperimentali effettuate utilizzando uno strumento atto allo scopo: il freno dinamometrico.

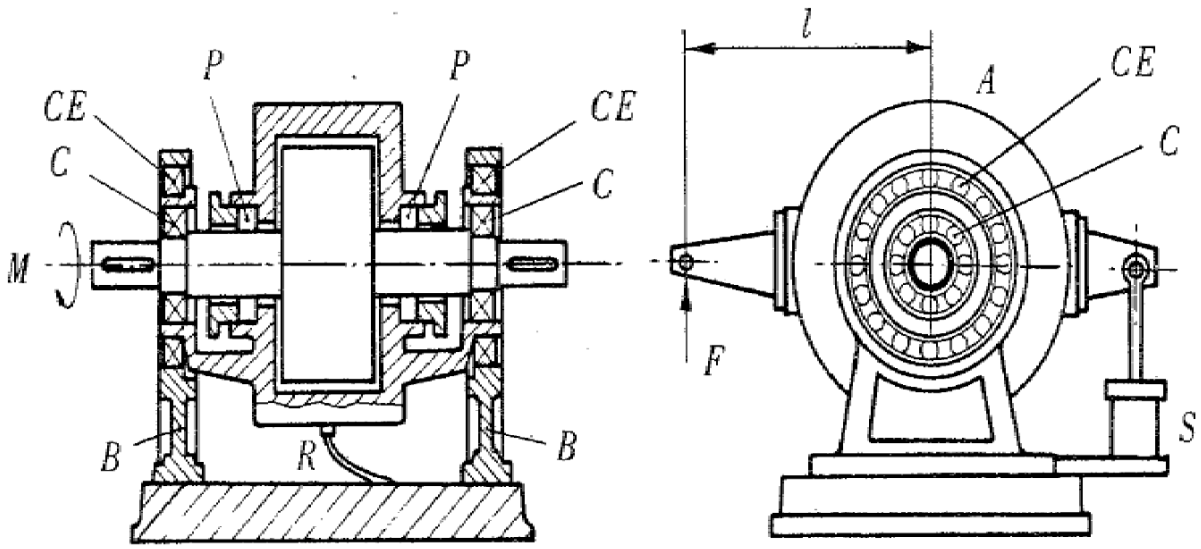



*Figura 2: Schema di funzionamento freno dinamometrico*

Esso ha il compito di dissipare l'energia meccanica prodotta dal motore simulando il collegamento con un utilizzatore. L'albero motore è accoppiato al rotore mediante forze di natura idraulica o elettromagnetica ad uno statore libero di muoversi. Il principio di funzionamento si basa sul fatto che il motore in prova tende a generare un campo di forze elettromagnetiche rotanti. Una cella di carico misura la forza necessaria per mantenere lo statore fermo. La forza misurata moltiplicata per il braccio tra l'asse dell'albero e l'asse della cella mi danno la coppia effettiva.

$$M_e = Fb \qquad (1.6)$$

Per il calcolo della potenza effettiva basta andare a moltiplicare la coppia effettiva per la velocità angolare:

$$P_e = 2\pi n M_e \qquad (1.7)$$

dove: $n$= numero di giri

Una volta calcolati questi due parametri sono utili per andare a vedere graficamente le prestazioni di un motore andando ad analizzare le curve caratteristiche. Le suddette curve permettono di analizzare la coppia e la potenza effettiva in funzione del regime di rotazione (Figura 3). Un appunto importante è da fare sul fatto che, andando a definire la coppia e la potenza solo in funzione del regime di rotazione non si riesce ad avere un valore univoco dei suddetti valori in quanto un parametro molto importante che influenza l'andamento delle curve è il carico. Normalmente le curve caratteristiche vengono realizzate andando a considerare la condizione di massima ammissione del carico.

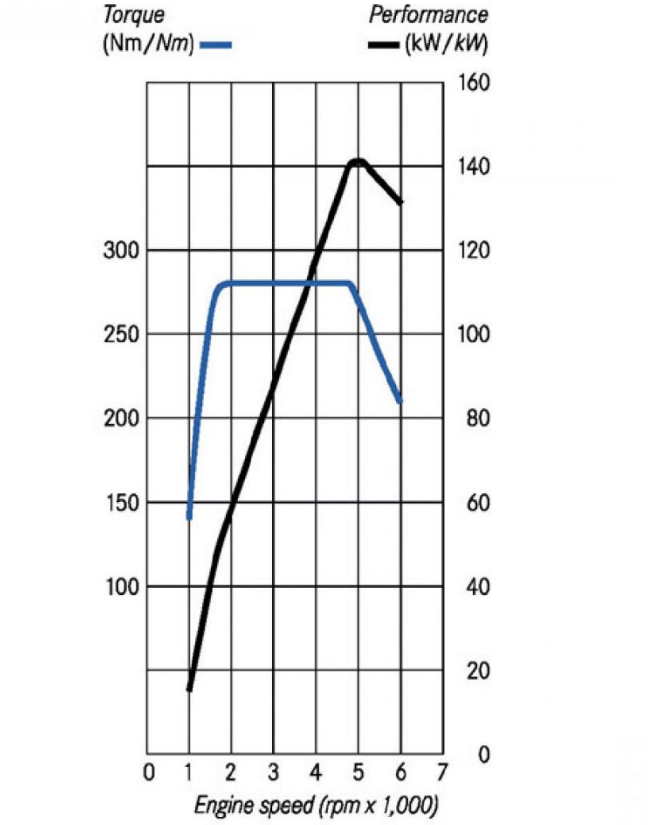



*Figura 3: Esempio curve caratteristiche di un motore ad accensione comandata sovralimentato*

## 1.3. Sviluppo della combustione

La combustione è un fenomeno molto importante per un motore in quanto è la causa primaria per la quale si ha rilascio di energia. È un fenomeno fisico/chimico molto complesso e di difficile interpretazione in quanto fortemente influenzato da svariati parametri. Il suddetto processo è definito come un fenomeno fisico e chimico perché l'ossidazione del combustibile in camera è prettamente un fenomeno chimico che però, per far sì che si sviluppi in modo corretto ha bisogno che combustibile e comburente siano ben miscelati e quindi c'è bisogno di movimentare il fluido e per questo lo definisco anche fisico. Un parametro che influenza fortemente la combustione, e quindi le prestazioni di un motore, riguarda l'autoaccensione. Questo parametro viene inteso come un insieme di reazioni di prefiamma le quali, liberando dei prodotti instabili di parziale ossidazione, hanno il compito di avviare il processo di combustione senza l'intervento di una sorgente di accensione esterna.
L'autoaccensione è fortemente dipendente da svariati fattori, come ad esempio la temperatura, pressione, composizione chimica del combustibile e da un tempo, chiamato tempo di autoaccensione che è una proprietà del materiale ed è il tempo che trascorre da quando la miscela, che è stata portata in determinate condizioni di pressione e temperatura, inizia la fase di combustione.
La combustione è quindi quel fenomeno che inizia dal momento in cui scocca la scintilla, parlando di un motore ad accensione comandata, e termina quando il combustile ha esaurito la sua energia ed è pronto per essere sostituito da carica fresca. Nel processo di combustione quindi si possono identificare più fasi che descrivono l'andamento temporale, preso in riferimento all'angolo di manovella, del fenomeno. Le fasi individuate sono tre e si dividono in:

1. fase di incubazione o sviluppo della fiamma: il primo nucleo di miscela brucia gradualmente
2. fase di combustione turbolenta: fase caratterizzata dalla rapida propagazione del fronte di fiamma, turbolento e pienamente sviluppato.
3. fase di completamento: questa fase è caratterizzata dal fronte di fiamma che raggiunge le pareti più lontane della camera fermandosi una volta raggiunta la parete stessa.

Una combustione viene definita normale, sempre parlando di un motore ad accensione comandata, quando ha inizio allo scoccare della scintilla e si propaga in maniera graduale senza avere brusche variazioni di velocità. Se queste condizioni non vengono verificate la combustione viene definita anomala. Le combustioni anomale si suddividono in due forme principali:

- accensione a superficie
- detonazione

L'accensione a superficie avviene quando la miscela viene accesa in un momento diverso da quello ottimale iniziando la propria combustione da un punto caldo presente in camera. Questo fenomeno si suddivide a sua volta in due fenomeni, la

preaccensione e la postaccensione, in base a quando si è avviata la combustione anomala, se prima o dopo lo scoccare della scintilla. Ovviamente il fenomeno di accensione a superficie è un fenomeno molto complesso che è determinato in primis dalla temperatura del punto caldo, poi dalla pressione, che deve essere tale da portare la miscela in condizioni di autoaccensione, e dalla composizione della miscela. Il fronte di fiamma che si sviluppa è molto simile a quello che si sviluppa quando avviene lo scoccare della scintilla. Per quanto riguarda i danni, essi posso essere di notevole importanza in quanto, se siamo in condizioni di preaccensione molto anticipata, la combustione potrebbe avvenire nella fase di compressione, ovvero nella fase in cui il lavoro è resistente e che potrebbe portare ad avere il lavoro motore molto inferiore al resistente, quindi avendo un lavoro utile nullo. Un altro problema molto importante che nasce con questo tipo di combustione anomala riguarda la temperatura e la pressione del punto caldo che tendono ad aumentare rendendo il processo incontrollabile e soprattutto autoesaltante che porta alla distruzione degli organi meccanici del motore

La detonazione, che è l'altra forma di combustione anomala, è un fenomeno potenzialmente molto dannoso per il motore stesso. Il fenomeno consiste nell'accensione di una parte di carica fresca non ancora raggiunta dal fronte di fiamma. In particolare, se si va a considerare la carica all'interno della camera, essa non sarà mai perfettamente omogenea e distribuita, stessa cosa che vale per pressione e temperatura soprattutto appena inizia la combustione. Una volta scoccata la scintilla si genererà all'interno del cilindro un fronte di fiamma che tende a comprimere e scaldare la restante parte di miscela contenuta nella camera. Ora considerando la natura non omogenea della carica potrebbe andarsi a creare una situazione in cui una parte della carica fresca non raggiunta dal fronte di fiamma si accenda localmente. Questo porta ad un picco di pressione che nella migliore delle ipotesi causa delle vibrazioni e del riscaldamento anomalo ma che nella peggiore delle ipotesi porta a rotture di organi importanti del motore come mostrato nella figura 4.

*Figura 4: Esempio di pistone danneggiato da detonazione*

## 1.4. Sviluppo della pressione nel cilindro

Il fenomeno della combustione genera un aumento di pressione nel cilindro che ci consente di compiere ulteriori analisi sulla combustione.

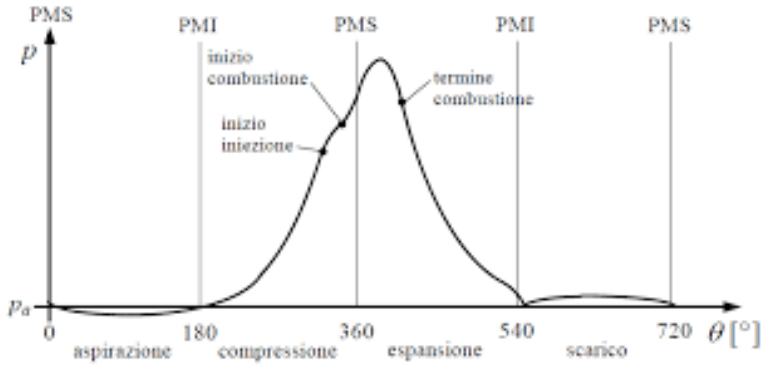



*Figura 5: Classico andamento pressione per motore ad accensione comandata*

La figura 5 ci presenta un classico andamento della pressione all'interno di un cilindro di un motore ad accensione comandata. L'importanza che riserviamo all'analisi di questa curva risiede nel fatto che dall'andamento della pressione nel tempo dipende il valore del lavoro meccanico raccolto all'albero motore, ovvero il lavoro positivo. Come già anticipato nel capitolo precedente, per avere una correlazione ottimale tra la posizione dell'albero motore e il lavoro generato, bisogna tener conto della natura non istantanea della combustione. Questo comporta un'analisi dettagliata sul momento preciso in cui far iniziare la fase di combustione. In particolare, per far sì che la pressione si sviluppi nel punto che garantisce il massimo lavoro positivo, si anticipa lo scoccare della scintilla prima del PMS di fine compressione in modo da avere il rilascio di energia nel momento a cavallo del PMS. Per determinare il momento ottimale si ricorre all'utilizzo di modelli di calcolo che poi vengono perfezionati in sala prova. Questo perché se si anticipa di tanto l'accensione una delle possibili conseguenze è che la miscela inizi la sua fase di combustione molto prima del PMS di fine compressione, andando a rilasciare il picco di pressione mentre il pistone sta ancora comprimendo, ciò non è per niente positivo visto che il lavoro svolto in fase di compressione è negativo e corrisponde al lavoro resistente, se invece si ritarda di molto l'accensione, il lavoro utile prodotto nella fase di espansione subisce una notevole diminuzione dovuta all'aumento di volume disponibile al fluido per espandersi.

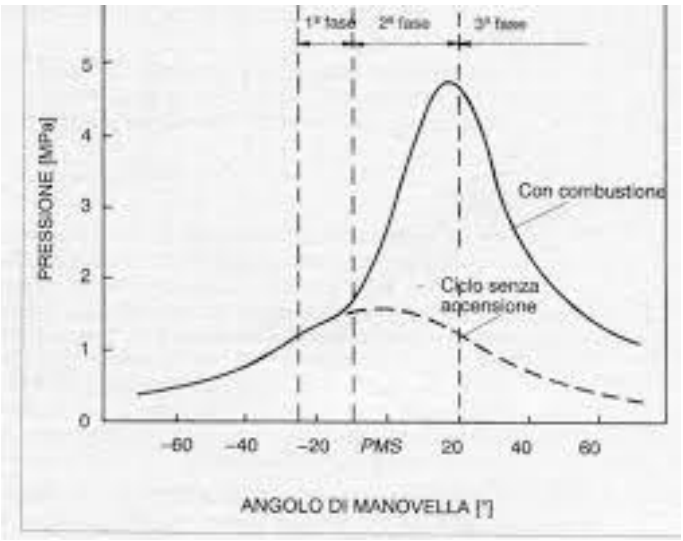



*Figura 6: Sviluppo di pressione nel cilindro riferito alle fasi della combustione*

La figura 6, invece, ci mostra come la pressione si sviluppa nel cilindro andando ad evidenziale le tre fasi di combustione. La premessa che è da fare riguarda il fatto che, ovviamente, i relativi istanti di separazione tra le varie fasi non possono essere determinati in modo preciso perché la velocita di combustione varia molto gradualmente. Detto questo, analizzando la curva, possiamo vedere come la pressione del cilindro partendo da -60° inizia gradualmente a crescere, questo è dovuto al fatto che il pistone si trova nella fase di compressione. Ci accorgiamo dell'inizio della fase di combustione quando la pressione tende ad aumentare rispetto alla legge di compressione, questo corrisponde alla fase 1. La fase 2 è facilmente riconoscibile in quanto essa è caratterizzata da un brusco aumento di pressione fino a raggiungere il suo picco e poi iniziare la fase di discesa che continuerà nella fase 3.

# 2

# SISTEMI DI ANALISI DELLA COMBUSTIONE

## 2.1. Controllo in feedback della combustione

Le stringenti normative sulle emissioni massime di inquinanti provenienti dalle fasi di combustione di un MCI e l'esigenza di avere dei motori con prestazioni ottimizzate spingono i progettisti ad avvalersi di sistemi elettronici per l'analisi della combustione.

Perché è interessante analizzare la combustione?

È interessante analizzare la combustione in quanto è la principale causa della formazione di inquinanti e soprattutto è il processo che converte l'energia rilasciata dall'ossidazione del combustibile in lavoro motore che può essere raccolto all'albero.

Per compiere un'analisi sulla combustione si ricorre all'utilizzo, indispensabile, dell'elettronica che è usata per andare a verificare ciclo per ciclo i vari parametri che vengono forniti come input. Poi da qui, mediante l'utilizzo di un sistema di gestione centralizzato, il sistema è capace di restituire degli output seguendo delle mappe opportunamente realizzate. Questo serve per andare a modificare i parametri fuori range del ciclo successivo.

Un classico sistema utilizzato per il controllo dell'iniezione di combustibile nei motori ad accensione comandata aspirati è:

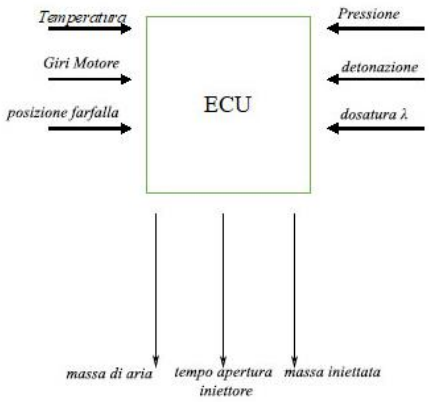



*Figura 7: Schema del processo di iniezione*

Come è possibile notare nella figura 7, in un motore ad accensione comandata aspirato, il sistema di gestione elettronico ECU (Electronic Control Unit) riceve i segnali di: temperatura, pressione, giri motore, detonazione, posizione farfalla, rapporto A/F e, mediante mappe memorizzate nella ECU, riesce a controllare la massa di aria iniettata e la quantità di combustibile da iniettare nel ciclo successivo. Questo sistema è quindi capace di attuare delle decisioni sugli output, previo però la conoscenza di mappe ottimizzate per la gestione di input/output.
Per studiare le mappe i progettisti si avvalgono di sistemi di analisi della combustione, anche denominati CAS (Combustion Analysis, System), che sono in grado di elaborare i segnali provenienti dai vari sensori presenti nel motore in modo da studiare, mediante l'utilizzo di sofware di elaborazione dati, il comportamento dei parametri importanti per la regolazione del motore.

## 2.2. Parametri analizzati e come calcolarli

### 2.2.1 Pressione Media Indicata e Pressione Media Effettiva

Uno dei parametri più importante da analizzare in un'analisi di combustione è la pressione. La misurazione avviene andando a disporre un trasduttore di pressione che si affaccia all'interno del cilindro. Questo dato, insieme alla misura del volume disponibile per il fluido evolvente mi consente di calcolare le grandezze indicate. Il nome di queste grandezze deriva dalla definizione di ciclo indicato, nome che deriva dai primi sistemi di acquisizione dati che venivano chiamati indicatori.
Una volta conosciuta la relazione che lega pressione e volume è possibile calcolare il lavoro indicato per ciclo. Quest'ultimo rappresenta il lavoro che il fluido cede al pistone e si calcola con la seguente formula:

$$L_i = \oint p d\boldsymbol{V} \qquad (2.2.1)$$

Per quanto riguarda invece la potenza indicata, ovvero la potenza ceduta dal fluido al pistone è calcolata come il prodotto del lavoro indicato per la frequenza di ciclo:

$$P_i = L_i f_c = L_i \, n/\varepsilon \qquad (2.2.2)$$

Dove il termine $f_c = n/\varepsilon$ sta ad indicare il numero di cicli per unità di tempo.
Il lavoro indicato definito nella 2.2.1 tiene conto anche della dimensione del cilindro. Ciò rende questo dato inutilizzabile per confrontare motori di varie geometrie e dimensioni. Per ovviare a questo problema si va ad eliminare la dipendenza dal volume andando a dividere il lavoro indicato per l'unità di cilindrata. Questa grandezza così generata prende il nome di $p_{mi}$, pressione media indicata, o in inglese IMEP. La relazione della $p_{mi}$, così come è stata definita è:

$$p_{mi} = L_i/V = (1/V) \oint p \, d\boldsymbol{V} \qquad (2.2.3)$$

La relazione così descritta rappresenta il rapporto tra l'area racchiusa dal ciclo indicato e la cilindrata spazzata. La $p_{mi}$ a livello geometrico può essere vista come l'ordinata media del ciclo indicato. È di notevole importanza andare a puntare l'attenzione sul fatto che, se la $p_{mi}$ agisse in maniera costante su tutta la corsa di espansione produrrebbe tutto il lavoro indicato. Questo è più chiaro se si analizza la relazione:

$$L = Fs = (p_{mi}A_c)C = p_{mi}V = L_i \qquad (2.2.4)$$

E richiamando la 2.2.2 si ottiene:

$$P_i = p_{mi}V \, n/\varepsilon \qquad (2.2.5)$$

Per compiere il passaggio dalle grandezze indicate a quelle effettive bisogna definire un parametro molto importante che racchiude in sé tutte le energie spese per vincere gli attriti tra gli elementi cinematici del motore. Il parametro che occorre descrivere è il rendimento organico $\eta_o$. Viene definito in termini di potenze e sta a rappresentare il rapporto tra la potenza effettivamente raccolta all'albero e la potenza indicata. La relazione è la seguente:

$$\eta_o = P_e/P_i \qquad (2.2.6)$$

una volta definito il rendimento organico è possibile andare a definire la potenza media effettiva $p_{me}$ che è definita come il lavoro effettivo per ciclo e unità di cilindrata. È in grado di farci capire l'efficienza con la quale il progettista è riuscito a sfruttare tutta la cilindrata del motore per un dato regime di rotazione. La relazione che ci esplica questa grandezza è:

$$p_{me} = \eta_o p_{mi} \qquad (2.2.7)$$

dopo aver stimato la pressione media effettiva è possibile andare a prevedere la potenza effettiva che possiamo raccogliere all'albero seguendo la relazione:

$$P_e = p_{me}V \, n/\varepsilon \qquad (2.2.8)$$

Queste relazioni vengono poi richiamate nei vari sistemi di analisi della combustione nel seguente modo:

$$IMEP = \int_{-360}^{360} p \frac{dV}{V} \qquad (2.2.9)$$

$$IMEPH = \int_{IVC}^{EVO} p \frac{dV}{V} \quad (2.2.10)$$

$$IMEPL = \int_{EVC}^{IVO} p \frac{dV}{V} \quad (2.2.11)$$

$$IndiTorque = IMEP \frac{V}{20\,\pi z} \quad (2.2.12)$$

dove:

- V = è la cilindrata
- p = è la pressione
- gli indici che compaiono nell'integrale della IMEP, -360-360, stanno ad indicare l'intero ciclo
- EVO = è l'acronimo di Exhaust Valve Opening, ovvero sta a rappresentare la posizione angolare in cui le valvole di scarico sono aperte
- IVC = è l'acronimo di Intake Valve Closing, ovvero sta ad indicare la posizione angolare in cui le valvole di aspirazione sono chiuse
- EVC = indica la posizione angolare in cui le valvole di scarico sono chiuse, Exhaust Valve Closing
- IVO = indica la posizione angolare in cui le valvole di aspirazione sono aperte, Intake Valve Opening
- IMEPH è la IMEP calcolata a valvole chiuse
- IMEPL è la IMEP calcolata a valvole aperte

## 2.2.2 Calore rilasciato

Per descrivere l'andamento del processo di combustione si possono usare vari parametri come la pressione, già vista nel paragrafo precedente, o la legge di rilascio dell'energia. Normalmente si delineano dei modelli di calcolo per andare a studiare il suddetto parametro. Un modello molto semplice di analisi si basa sul primo principio della termodinamica applicato al sistema gassoso contenuto nella camera di combustione. Un'approssimazione che va fatta in questa analisi è andare a considerare il volume gassoso considerato come omogeneo, anche se la combustione è in corso. L'energia liberata dalle reazioni chimiche in funzione dell'angolo di manovella $\theta$ ($\mathrm{d}Q_b/d\theta$) unito al passaggio di materia attraverso la superficie di controllo ($\sum_i h_i dm_i/d\theta$) deve essere uguale all'aumento di energia interna del sistema, sempre riferito all'angolo di manovella, ($\mathrm{dE}/\mathrm{d}\theta$) unito con lo scambio di potenza meccanica con l'esterno attraverso il pistone ($\mathrm{dL}/\mathrm{d}\theta$) unito con le perdite di calore dovute al contatto del fluido con le pareti refrigerate della camera di combustione ($\mathrm{d}Q_r/d\theta$). Unendo tutte queste considerazioni la relazione diventa:

$$\frac{dQ_b}{d\theta} + \sum_i h_i \frac{dm_i}{d\theta} = \frac{dE}{d\theta} + \frac{dL}{d\theta} + \frac{dQ_r}{d\theta} \quad (2.2.13)$$

Nel caso di un motore a ciclo Otto in cui, normalmente, non c'è iniezione di combustibile durante il processo di combustione e quindi attraverso le superfici del volume di controllo, la quantità riferita agli scambi di massa tra volume e ambiente esterno, il secondo termine a destra dell'uguale della 2.2.13, viene ritenuto uguale a zero, perché considerato il rapporto $dm_i/d\theta$ nullo. Se inoltre consideriamo il fluido come gas perfetto a temperatura media T, allora l'energia interna può essere considerata E=m$c_v$T, e la sua derivata rispetto all'angolo di manovella viene:

$$\frac{dE}{d\theta} = mc_v \frac{dT}{d\theta} + c_v T \frac{dm}{d\theta} \qquad (2.2.14)$$

in cui, ovviamente, se si sta considerando nullo lo scambio di masse, il secondo termine dopo l'uguale diviene trascurabile in quanto dm=0.
La potenza raccolta dal pistone, invece, può essere indicata come:

$$dL/d\theta = pdV/d\theta \qquad (2.2.15)$$

mentre esprimendo la temperatura in funzione di pressione e volume e andando a differenziare rispetto all'angolo di manovella si ottiene:

$$T = pV/mR \qquad (2.2.16)$$

$$\frac{dT}{d\theta} = \frac{p}{mR}\frac{dV}{d\theta} + \frac{V}{mR}\frac{dp}{d\theta} \qquad (2.217)$$

Da qui, andando a tener conto delle equazioni elencate in precedenza, la legge di rilascio dell'energia, 2.2.13, diventa:

$$\frac{dQ_b}{d\theta} = \frac{k}{k-1} p \frac{dV}{d\theta} + \frac{1}{k-1} V \frac{dp}{d\theta} + \frac{dQ_r}{d\theta} \qquad (2.2.18)$$

dove: k=$c_p/c_v$ e pressione e volume sono funzioni dell'angolo di manovella $\theta$
Integrando la 2.2.18 tra l'angolo di inizio combustione e l'angolo di fine combustione, riesco a calcolare tutta l'energia liberata, che è circa uguale alla massa di combustibile iniettato moltiplicata per il suo calore specifico.

$$Q_b = \int_{\theta_i}^{\theta_f} (dQ_b/d\theta) d\theta \simeq m_c H_i \qquad (2.2.19)$$

La curva che descrive l'andamento della legge di rilascio dell'energia è chiamata ROHR, Rate Of Heat Release, ed è rappresentata in figura 8.

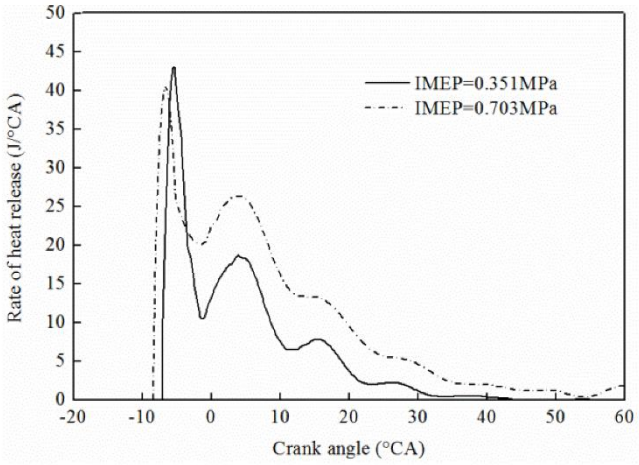
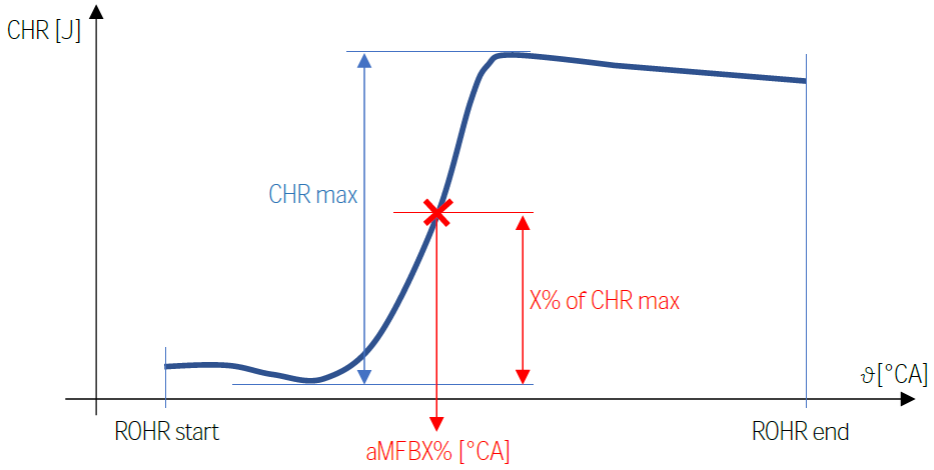



*Figura 8: Esempio curva di rilascio energia ROHR*

L'equazione utilizzata dai sistemi di analisi della combustione per il calcolo del CHR, Cumulative net Heat Release, che sta a rappresentare la quantità di energia liberata durante la combustione, è la seguente:

$$CHR = \int_{ROHR\ begin}^{ROHR\ end} \frac{1}{k-1} Vdp + \frac{k}{k-1} pdV \qquad (2.2.20)$$

dove: k = $c_p/c_v$ per la fase di compressione [ROHR begin; 0], mentre k = $c_v/\ c_p$ nella fase di espansione [0; ROHR end].
La curva che descrive l'andamento del CHR in relazione all'angolo di manovella è invece espressa nella figura 9.




$$aMFBX\% = angle\ position\ of\ X\%\ of\ CHR\ MAX$$

*Figura 9: Curva del CHR in funzione dell'angolo di manovella*

La figura 9 pone l'attenzione su un parametro molto importante nell'analisi della combustione che è l'MFBx, dove la x sta ad indicare una percentuale riferita allo stato della combustione. In effetti, a causa delle stringenti normative per il controllo

delle emissioni inquinanti, i sistemi a circuito chiuso richiedono la valutazione dell'MFB50. Questo parametro mi dice in che posizione angolare avrò bruciato il 50% del combustibile iniettato. Normalmente nell'analisi della combustione si vanno ad indicare diversi valori di x, in modo da avere una descrizione più dettagliata dell'evoluzione della combustione riferita alla posizione angolare.

### 2.2.3 Indice di detonazione

Un indice da tenere sotto controllo nell'analisi della combustione è l'indice ci detonazione, o in inglese knock index. Questo indice è di notevole importanza per non indurre a rotture nel motore. Come già annunciato nel capitolo 1.3, in cui si va a descrivere il fenomeno della combustione e si vanno ad individuare le combustioni anomale, se un motore inizia a detonare e il sistema non è capace di controllare il fenomeno i danni causati dalla detonazione possono essere di notevole importanza fino a compromettere la funzionalità stessa del motore. Un sistema per prevenire danni ingenti è quello in cui si va a vedere ciclo per ciclo se c'è detonazione in modo da correggere i parametri della combustione del ciclo successivo. Un metodo per andare a calcolare l'indice di detonazione è il MAPO. L'indice restituito da questo metodo è semplicemente la massima ampiezza del modulo del segnale della pressione, filtrato passa alto, che viene rilevato in un determinato intervallo di tempo. La formula utilizzata per il metodo MAPO è la seguente:

$$MAPO = MAX(\left|HighPass(signal_{\theta_{start}}^{\theta_{end}}\right|) \qquad (2.2.21)$$

L'altro modo per andare a calcolare l'indice di detonazione è chiamato KINT, Knock Integral index. È l'integrale del segnale di pressione prelevato dal trasduttore filtrato passa alto e integrato nell'intervallo indicato. La formula utilizzata per calcolare l'indice usando questo metodo è:

$$KINT = \frac{\int_{\theta_{start}}^{\theta_{end}} |HighPass(signal)| d\theta}{number\ of\ samples\ in\ [\theta_{start};\theta_{end}]} \qquad (2.2.22)$$

### 2.2.4 Tasso di aumento del picco

Il tasso di aumento del picco della pressione, in inglese Peak Rise Rate, mi dice in che modo sta aumentando la pressione e soprattutto con che velocità sta crescendo. La formula che viene usata per calcolare il picco è:

$$Rmax = MAX\left(\frac{dP}{d\theta}|_{ROHR_{start}}^{ROHR_{end}}\right) \qquad (2.2.23)$$

Il valore Rmax è calcolato, quindi, come il picco massimo della derivata della pressione presa tra due intervalli ben precisi, che in questo caso stanno ad indicare l'inizio e la fine del rilascio dell'energia dalla combustione.
L'altro parametro che va a calcolare questa funzione è la posizione del valore Rmax, che viene semplicemente chiamata Amax.

## 2.3. Sistemi di analisi della combustione

I sistemi di analisi della combustione, o anche chiamati CAS, Combustion Analysis System, sono dei sistemi di acquisizione dati ad alte prestazioni progettati specificatamente per l'analisi della combustione di motori a combustione interna a due e quattro tempi alimentati sia a benzina che a diesel. Questi sistemi consentono di acquisire dati ad altissima velocità e soprattutto in tempo reale. I CAS digitalizzano dei segnali di pressione provenienti da trasduttori piezoelettrici posizionati nel cilindro e sincronizza i dati con un encoder montato sull'albero motore o un sensore di posizione in grado di restituire come output l'angolo di manovella che ci consente di capire in che posizione si trova l'albero motore e di conseguenza il pistone. Una volta elaborati questi segnali il sistema è capace di calcolare svariati parametri alcuni dei quali sono elencati nei paragrafi precedenti con le rispettive formule utilizzate per il calcolo. I sensori per misurare la pressione più comunemente utilizzati sono dei sensori piezoelettrici che, sottoposti ad un carico meccanico, come la pressione nella camera, in input, in output restituiscono un segnale di carattere elettrico. I sensori piezoelettrici sono ottimi per l'utilizzo in applicazioni dinamiche possono essere impiegati per misurare pressioni che cambiano in maniera molto rapida, e sono molto resistenti a temperature abbastanza elevate, circa 300°/400°. La parte più importante della struttura del sensore, il nucleo, è costituito da un cristallo. Il materiale più utilizzato è il quarzo in quanto è in grado di garantire un'elevata sensibilità e stabilità alla variazione di temperatura. I sensori di questo tipo si differenziano in base al loro utilizzo e soprattutto in base alle caratteristiche fisiche del motore. La differenza sta nella possibilità o meno di modificare la testa del motore. Se la testa può essere modificata, mediante trapanatura o fresatura, allora si può utilizzare un semplice sensore avvitato nell'alloggiamento creato. Se invece la testa non può essere modificata bisogna necessariamente ricorrere a fori già esistenti, uno dei quali potrebbe essere il foro della candela, andando ad utilizzare una candela strumentata. (figura 11)

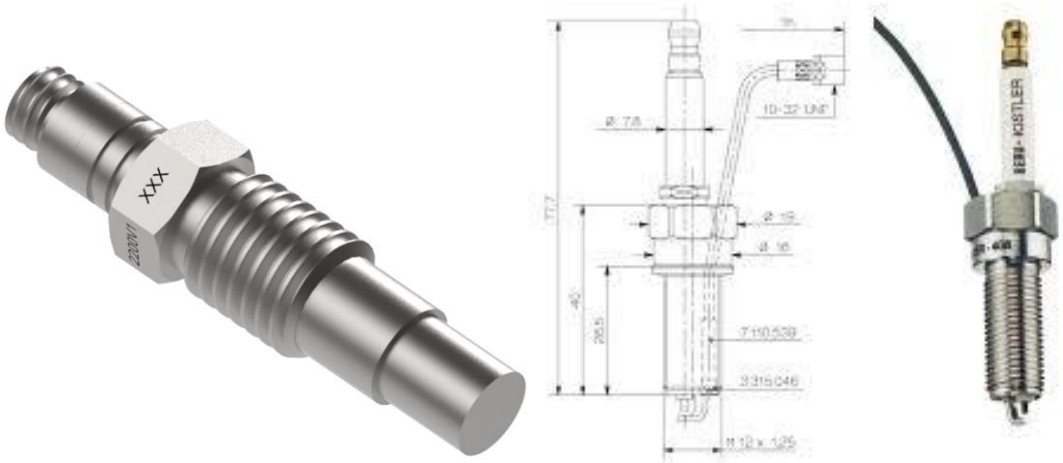



*Figura 10: A sinistra, classico esempio di sensore piezoelettrico. A destra esempio di candela strumentata prodotta da Kistler.*

I sensori che invece vanno a verificare la posizione dell'angolo di manovella sono di tipo encoder. Il principio che sta alla base è quello di andare a calcolare la posizione angolare sfruttando un rotore che ruota insieme all'albero al quale è collegato un disco/ingranaggio dentato o comunque costituito da un'alternanza di spazi vuoti e spazi pieni uniti ad un sensore capace di leggere la differenza tra pieno e vuoto. Gli encoder possono essere di diverso tipo. Encoder capacitivi/induttivi, sfruttano la capacità di un sensore di prossimità di leggere i denti dell'ingranaggio, denominato ruota fonica, o le forature di un disco. Altri tipi di encoder possono essere: encoder magnetici, si basano sull'effetto hall, encoder potenziometrici, in cui il sensore è un potenziometro che sfrutta la capacità di emettere un segnale elettrico proporzionale alla posizione assunta. L'ultima tipologia di encoder riguarda quelli ottici. Il funzionamento di questi encoder è basato su dei sensori ottici capaci di leggere una matrice di aree trasparenti e opache stampate sul rotore.

### 2.4.1 Principali CAS in commercio

I principali sistemi per l'analisi della combustione che troviamo in commercio sono:

- OBI-M2 di Alma Automotive
- KiBox di Kistler
- Indimaster di AVL

Le caratteristiche principali di questi sistemi di analisi della combustione sono la loro versatilità, infatti possono essere utilizzati sia per prove sul banco motore e sia a bordo del veicolo stesso. Il loro impiego può essere di molteplice interesse. Infatti, ad esempio, OBI-M2 è in grado non solo di ottimizzare la taratura del motore ma anche di rilevare eventuali superamenti di valori di soglia limite di alcuni parametri definiti dall'utente in modo da compiere anche il monitoraggio di parametri molto importanti.

# 3

# FONDAMENTI DEL LINGUAGGIO PYTHON

## 3.1. Introduzione a Python

Python nasce nel dicembre del 1989 per opera dell'informatico olandese Guido van Rossum. L'informatico aveva già, in passato, lavorato alla realizzazione di un linguaggio di programmazione che però non ebbe un consolidato successo da cui però rimase affascinato per alcuni aspetti che poi saranno le basi del nuovo linguaggio di programmazione. Su queste basi van Rossum iniziò la scrittura in C di un nuovo linguaggio di programmazione che poi battezzerà con il nome di Python in onore alla sua serie televisiva preferita: Monty Python's Flying Circus.
Python è sviluppato, mantenuto e rilasciato da un gruppo di persone a cui fa capo proprio l'informatico olandese e che nel 2001 fondò la Python Software Foundation (PSF), associane not-for-profit che detiene il diritto d'autore e ne promuove la diffusione. La nomenclatura delle versioni di Python si suddivide in major version, minor version e micro number. Normalmente le major version vengono rilasciate a distanza di diversi anni e costituiscono il primo numero nell'identificazione della versione. Mentre le minor version vengono rilasciate con cadenza annuale. Un aspetto molto importante che riguarda le versioni è che ogni programma scritto in una determinata minor version è completamente compatibile con le successive minor version, ciò invece non è assicurato con le major version.
Python è un linguaggio molto robusto e maturo utilizzato in tantissimi ambiti che vanno dal web allo sviluppo di interfacce grafiche, programmazione di sistema e programmazione di giochi e multimedia. La sua versatilità di utilizzo fa sì che esso sia costantemente utilizzato da grandi aziende come Google, YouTube, Intel, Yahoo!, Spotify e Dropbox.
L'Italia gioca un ruolo molto importante nel panorama mondiale di Python, infatti dal 2011 al 2013 la Conferenza Europea di Python (EuroPython) si è tenuta proprio a Firenze. Inoltre, in Italia risiede una delle più grandi organizzazioni legate a Python, l'Associazione di Promozione Sociale Python Italia.

## 3.2. Il linguaggio di programmazione

Per scrivere un programma per calcolatore, occorre delineare una sequenza di istruzioni che possano essere poi lette ed eseguite dalla CPU. Un programma, quindi, non è nient'altro che un insieme di linee scritte in una sintassi particolare che poi vengono tradotte in istruzioni semplici che possono essere eseguite dalla CPU. Un linguaggio che è in grado di compiere questi passaggi è definito linguaggio di programmazione di alto livello. L'obiettivo dell'ideatore di Python era quello di creare un linguaggio di programmazione in cui sia facile e veloce andare a modificare i programmi in caso di errori importanti. Python si differenzia dai principali linguaggi di programmazione, come C, C++ e Java, per la sua estrema facilità nella scrittura e nella sua sintassi più semplice e chiara. Inoltre, Python ha un ambiente interattivo in cui è possibile andare a compilare piccoli spezzoni di codice e capire il loro comportamento rendendo più semplice il passaggio tra un'ambiente di programmazione e l'altro. Un altro grande vantaggio di Python è che, essendo open source, mette a disposizione un numero smisurato di moduli realizzati da programmatori professionisti che possono essere usati per la risoluzione di problemi specifici già noti.
In Python un ruolo molto importante è rivestito dall'indentazione del programma. Infatti, un blocco di codice nidificato non è delimitato da parole chiave, è identificato solo dal simbolo dei due punti e dall'indentazione stessa del codice (esempio codice 1)

```
1  for i in range (2):
2      if i % 2 == 0
3          print ("Sono all''interno del blocco if")
4          print (i, "è un numero pari.\n")
5          continue
6      print ("il blocco if è stato saltato")
```

*Esempio codice 1*

Per blocco nidificato si intendono tutte quelle istruzioni precedute dal simbolo dei due punti. Detto ciò, l'indentazione diventa una prerogativa al funzionamento dello script e non una questione di stile. Questo perché la 'regola' base sulla quale si basa la programmazione in Python è riuscire ad eseguire una determinata condizione seguendo un modo ovvio e preferibilmente unico.
Quando un programma viene eseguito, Python, a partire dal codice, genera delle strutture dati, chiamati oggetti, sulle quali basa poi tutto il processo di elaborazione. Gli oggetti vengono tenuti nella memoria del computer, in modo da poter essere richiamati quando il programma fa riferimento a essi. Nel momento in cui non servono più, un particolare meccanismo, chiamato *garbage collector*, provvede a liberare la memoria da essi occupata. Gli oggetti che costituiscono il cuore di Python, detti oggetti *built-in*, vengono comunemente distinti nelle seguenti categorie: *core data type*, *funzioni built-in*, classi e tipi di *eccezioni built-in*. Ciò che viene definito come *core data type* è l'insieme dei principali tipi built-in. Questi possono essere raggruppati in quattro categorie:

- numeri: interi (tipo `int`), booleani (tipo `bool`), complessi (`complex`), floating point (tipo `float`);
- insiemi: rappresentati dal tipo `set`;
- sequenze: stringe (`str` e `byte`), liste (`list`) e tuple (`tuple`);
- dizionari: rappresentati dal tipo `dict`.

I tipi del core data appartengono a una categoria di oggetti chiamati *classi*, o anche *tipi*. La caratteristica principale è quella di rappresentare un tipo di dato generico, dal quale poter creare oggetti specifici di quel tipo, chiamati *istanti*. Ad esempio, dal tipo `str` possiamo creare le istanze `"python"`, `"Guido"` e `"abc"`, dal tipo `int` istanze `22,77`, dal tipo `list` le istanze `[1, 2, 3]` e `['a', 'b', 'c', 'd']`. Quindi diremo che una stringa di testo è un oggetto di tipo `str`, o diremo che un intero è un oggetto di tipo `int`. Per identificare la tipologia di un oggetto basta passarlo come argomento della funzione built-in `type`. Gli oggetti sono sempre caratterizzati da un tipo, mentre solo ad alcuni di essi possiamo associare in modo intuitivo un valore. Oltre al tipo, altri elementi caratteristici degli oggetti sono l'*identità* e gli *attributi*. L'identità è rappresentata da un numero che li identifica in modo unico, e viene restituita dalla funzione built-in `id()`. Gli attributi, invece, sono degli identificativi accessibili per mezzo del delimitatore punto. Essi sono fortemente legati al tipo di un oggetto. Ad esempio tutti gli oggetti di tipo `str` hanno l'attributo `str.upper()` che restituisce una versione maiuscola della stringa, mentre gli oggetti di tipo `list` hanno l'attributo `list.sort()` che riordina gli elementi della lista. Se un identificativo può essere seguito dalle parentesi tonde, si dice che l'oggetto è chiamabile (collable). Quando applichiamo le parentesi tonde all'identificativo diciamo che stiamo *chiamando* l'oggetto. Cosa molto importante da ricordare è che, anche se non dobbiamo passare degli argomenti, se vogliamo chiamare un determinato oggetto bisogna comunque usare le parentesi tonde. Le parentesi, infatti, indicano che vogliamo eseguire le operazioni che competono all'oggetto chiamabile. Gli attributi chiamabili sono detti *metodi*. La differenza principale tra i metodi e gli attributi è che i primi possono essere chiamati per eseguire delle operazioni, mentre i secondi no.
Le stringhe di testo in Python sono rappresentate da una sequenza di caratteri Unicode di lunghezza arbitraria, racchiusi tra apici singoli, virgolette, tripli apici singoli o triple virgolette:

```
1 s1 = 'stringa di testo' #Non può essere spezzato su più linee
2 s2 = "stringa di testo" #Non può essere spezzato su più linee
3 s3 = '''stringa
4 di testo''' #può essere spezzato in più linee
5 s4 = """stringa
6 di testo""" #può essere spezzato in più linee
```

*Esempio codice 2: esempio stringhe*

Anche le *liste* appartengono alla categoria delle sequenze, ma, a differenza delle stringhe, possono contenere oggetti di qualunque tipo. Vengono rappresentate inserendo gli elementi tra parantesi quadre, separati con una virgola:

```
mylist = ['ciao', 100, 33] # la lista contiene gli elemnti ciao, 100, 300
mylist.index(100)
1
mylist[1] #Indicizzazione
100
mylist[0:2]
['ciao', 100]
'ciao' in mylist #L'elemto 'ciao' è contenuto il mylist? (restituisce True o False)
True
```

*Esempio codice 3: esempio liste*

le liste possono essere modificate e quindi vengono chiamati oggetti *mutabili.*
Altri oggetti analoghi alle liste sono le *tuple*. Anch'esse appartengono alla categoria delle sequenze e possono contenere oggetti di qualunque tipo, ma, a differenza delle liste, sono immutabili, e vengono rappresentate inserendo gli elementi tra parentesi tonde piuttosto che tra parentesi quadre:

```
t = (1, 'due', [1, 'lista'], 'due', ())
t.count('due') #Resituisce il numero di occorrenze dell'elemento 'due'
2
t[1]
'due'
```

*Esempio codice 4: esempio tuple*

I dizionari, invece, sono dei contenitori aventi per elementi delle coppie *chiave:valore*. Vengono rappresentati inserendo gli elementi tra parentesi graffe, separandoli con una virgola:

```
d = {'cane':'bau', 'gatto':'miao', 'uno':1,2:'due'}
d['cane']
'bau'
```

*Esempio codice 5: esempio dizionario*

a differenza degli oggetti appartenenti alla categoria delle sequenze, i dizionari non sono dei contenitori ordinati, per cui la ricerca va fatta per chiave e non per indice, come mostrato nell'esempio codice 5.

## 3.3. L'ambiente di programmazione

Una distinzione doverosa da fare sta proprio sul termine Python che spesso viene erroneamente utilizzato per indicare due cose strettamente correlate: il linguaggio Python e l'interprete Python. Il linguaggio Python è quell'insieme di sintassi dedicata che permette di scrivere un codice facilmente comprensibile dalla macchina. È una lingua a tutti gli effetti che è possibile assimilare all'italiano o all'inglese.
L'interprete di Python, invece, è lo strumento capace di analizzare, capire ed eseguire il programma scritto nel linguaggio Python. L'interprete, in base al sistema operativo utilizzato, può essere già integrato, come nel caso dei sistemi Linux, oppure c'è bisogno di scaricare un IDLE dedicato, come nel caso di Windows che permette di andare a compilare il programma in .py. Per generare il codice in Python c'è bisogno prima di scriverlo nella sintassi dedicata. Questo può avvenire mediante l'utilizzo di un editor di testo già contenuto nel sistema operativo, oppure può essere richiamato dall'IDLE aprendo una nuova shell. Una volta completata la stesura del programma, esso può essere tranquillamente eseguito.

## 3.4. Architettura di un programma

Un programma scritto nel linguaggio Python contiene una o più righe di istruzioni o enunciati, che verranno poi tradotti ed eseguiti dall'interprete Python. In Python un programma è costituito da moduli, i quali contengono una serie di istruzioni le quali processano delle espressioni. In un programma di Python gli oggetti vengono creati dal codice in esecuzione. Adesso andiamo ad individuare la porzione di codice che genera gli oggetti oppure che semplicemente fa riferimento ad essi. La porzione che genera l'oggetto è chiamata *espressione* che è definita come quella porzione di codice contenuta in una linea logica, che quando il programma è in esecuzione fa riferimento a un oggetto oppure lo genera. Le espressioni si trovano alla destra del simbolo di uguale e sono confinate all'interno delle linee logiche. Per scoprire se una porzione di codice è una espressione possiamo estrapolarla dalla linea logica e assegnarla a una etichetta, che definiremo nel capitolo successivo. Se l'assegnamento non dà luogo a errori allora la porzione di codice è un'espressione.
In Python il codice che definisce una linea logica nella sua interezza è detto *istruzione semplice (simple statement)*, mentre un blocco di codice composto da istruzioni semplici è detto *istruzione composta (compound statement)*. Le istruzioni composte si caratterizzano per l'utilizzo del delimitatore dei due punti il quale indica l'inizio del blocco di codice dell'istruzione. L'unione delle istruzioni semplici e di quelle composte costituisce l'insieme delle istruzioni di Python che hanno un nome che deriva dalle parole chiave utilizzate al loro interno.
Una linea logica costituita da una espressione è una istruzione semplice. Questo è l'unico caso di una istruzione che è anche un'espressione. Il risultato non può essere salvato, per questo motivo un'espressione viene utilizzata come istruzione solo in due casi. Il primo è quando si utilizza la modalità interattiva poiché, nonostante il risultato dell'espressione non venga salvato, esso viene scritto nello standard output e lo si può verificare visivamente. L'altra situazione è quella delle chiamate di funzioni e metodi che non restituiscono esplicitamente un oggetto poiché in questi casi interessa solo che vengano eseguite delle azioni e non c'è alcun risultato da salvare. Tutte le altre istruzioni semplici non sono delle espressioni e quindi non possono essere assegnate a una etichetta.
L'assegnamento è un'istruzione semplice che assegna l'oggetto generato dall'espressione alla destra del simbolo di uguale all'etichetta alla sinistra del simbolo stesso.

```
1 a = [1, 2, 3, 5] # Istruzione di assegnamento
2 a
3 [1, 2, 3, 5]
4 x = (a=[1, 2, 3, 5]) #L'istruzione di assegnemento non è una espressione
5 # Restituisce un errore di sintasso riferito all'uguale prima delle quadre
```

*Esempio codice 6: Assegnamento*

È possibile assegnare il medesimo oggetto a diverse etichette in un'unica soluzione. È anche possibile effettuare degli assegnamenti in parallelo, mediante lo spacchettamento.

Una parola chiave che riveste un ruolo importante, soprattutto per quanto riguarda la sintassi riservata ai cicli, è la parola `pass`. È un'istruzione semplice che non esegue nulla. Quando l'interprete la incontra nell'esecuzione del ciclo viene saltata e l'esecuzione prosegue dall'istruzione successiva.
L'istruzione `import` serve invece per importare i moduli e le librerie riservate.

```
1 import math
2 math.pi
```

*Esempio codice 7: esempio istruzione import*

le linee di codice indicare nell'esempio codice 7 producono come risultato l'importazione della libreria math e il calcolo del pigreco.
L'istruzione `from` è utilizzata per importante degli attributi contenuti nel modulo.
L'enunciato `print()` ha invece il compito di 'stampare' a schermo ciò che gli viene passato come argomento che può essere costituito da semplici variabili, numeri, stringhe, etc.

## 3.5. Elementi del codice Python

Analizzando un codice Python possiamo renderci conto che è composto da una serie di linee chiamate *linee fisiche*. Per l'interprete Python le linee fisiche non hanno nessun tipo di significato ma servono semplicemente per organizzare il codice in modo leggibile. Una linea logica, invece, normalmente corrisponde ad una linea fisica, ma, volendo, è possibile suddividere una linea fisica in più linee logiche, separando queste con un carattere di punto e virgola. È anche possibile ripartire una linea logica su più linee fisiche. Questo lo si vede spesso e lo si può fare sia utilizzando il carattere blackslash \, sia usando le parentesi tonde, come evidenziato nell'esempio 8.

```
1  a = 10 + 20 + \ #Divisione linee fisiche con \
2  30 + 40 + \
3  50
4  a
5  150
6  a = (10 + 20 + #Divisione linee fisiche con le tonde
7       30 + 40 +
8       50)
9  a
10 150
```

*Esempio codice 8: Esempio divisione linea fisica*

Il testo contenuto all'interno di una linea logica appartiene a una di queste categorie: *commenti, letterali, operatori, parole chiave, etichette* e *delimitatori.*
I commenti servono per andare a spiegare in maniera più esaustiva il funzionamento di alcune linee logiche di particolare importanza in modo da rendere più comprensibile il funzionamento di un programma. I commenti sono delle porzioni di testo che iniziano con il carattere cancelletto, #, e terminano con la fine della linea fisica. Questo comando non viene letto come linea logica perché serve al programmatore per rendere più comprensibile una determinata operazione spiegando il suo funzionamento.

I letterali sono invece delle rappresentazioni testuali che sono in grado di determinare facilmente e in modo univoco sia il tipo sia il valore di un oggetto. Questo perché le istanze dei tipi del core data type sono distinguibili le une dalle altre per il tipo e per il valore. Siccome queste istanze rappresentano il cuore del linguaggio Python è stato necessario associare a queste i letterali. Ad esempio, un *letterale stringa* di testo è una porzione di codice che troveremo delimitato da apici singoli, virgolette, tripli apici singoli o triple virgolette. Una volta iniziata la compilazione del programma, l'interprete Python andrà a creare dal letterale un oggetto di tipo `str` in cui andrà ad inserire in memoria il testuale indicato tra apici.
Gli operatori in Python sono dei simboli ai quali viene associato un determinato significato in relazione agli argomenti che gli vengono passati come operandi. Ad esempio, il simbolo + è un operatore che esegue la somma se gli operandi sono numeri, mentre effettua la concatenazione se gli operandi sono delle sequenze.
Le parole chiave, che nel linguaggio troveremo indicate come keywords, sono invece quella parte di sintassi dedicata in cui si fa corrispondere a determinate parole delle azioni ben specifiche. Non è possibile cambiare il significato delle keywords in quanto se andassimo a cambiare il loro significato l'interprete non sarebbe più in grado di comprendere quella determinata azione, e non possono essere seguite dal simbolo di =. Le keywords, quindi, non possono essere assegnate come delle variabili.

```
Here is a list of the Python keywords.  Enter any keyword to get more help.

False               class               from                or
None                continue            global              pass
True                def                 if                  raise
and                 del                 import              return
as                  elif                in                  try
assert              else                is                  while
async               except              lambda              with
await               finally             nonlocal            yield
break               for                 not
```

*Esempio codice 9: Elenco keywords*

nell'esempio di codice 9, troviamo indicato il risultato che si ha se si fa compilare dall'interprete la porzione di codice `help('keywords')`. Il risultato ci dà l'elenco delle keywords riservate. Le parole chiave si possono suddividere in due categorie e si suddividono in: parole chiave in cui la prima lettera è maiuscola, e a questa tipologia appartengono le parole `False`, `True` e `None`, e parole in cui la prima lettera è minuscola, a cui appartengono tutte le altre. Un'altra distinzione importante va fatta sul comportamento che ha il compilatore quando incontra le parole chiave. Infatti, se la parola chiave ha la prima lettera maiuscola gli viene associato un oggetto in memoria. Ad esempio, a True corrisponde un oggetto interpretato con il numero uno. Alle altre, invece, non viene assegnato nessun oggetto in memoria ma semplicemente vengono eseguite delle operazioni specifiche.
Nella sintassi di Python un ruolo molto importante è svolto dai delimitatori. Essi sono costituiti da delle sequenze di uno, due o tre caratteri che hanno il compito principale di delimitare o separare delle porzioni di codice. Un esempio può essere fatto con il simbolo dei due punti, :, che indica l'inizio di un blocco di codice. Oppure un altro

esempio può essere fatto con le parentesi tonde, che delimitano l'assegnazione di una variabile.
Tutto ciò che non può essere definito dalle precedenti categorie rientra nella macrocategoria delle etichette. La caratteristica principale è che devono essere definite prima di poter essere utilizzate perché il compilatore, una volta iniziata l'esecuzione del programma, fa corrispondere ad esse un oggetto già precedentemente assegnato. Fanno eccezione delle funzioni, definite `builtin_function_or_method`, in cui non c'è bisogno di definire un determinato argomento perché è un'etichetta già definita nel linguaggio, come la funzione `print`.
La principale differenza che risiede tra le etichette, i letterali e le parole chiave è che solo le etichette possono comparire alla sinistra del simbolo di uguale.

## 3.6. Enunciato '*if*'

L'enunciato `if` va a verificare quando un'istruzione è verificata, se è verificata viene eseguito un certo insieme di enunciati, altrimenti ne viene eseguito un altro.
L'istruzione `if` ha le clausole `elif` ed `else`. Un esempio di istruzione decisionale è indicato nell'esempio 10, in cui si cerca di replicare le azioni eseguite da un computer di controllo degli ascensori in cui manca il piano 'tredici'. Il programma, in particolare, va a verificare se il piano selezionato è maggiore di 13. In tal caso il piano che deve essere raggiunto dall'ascensore corrisponderà al piano selezionato meno 1, altrimenti il piano corrisponde al piano selezionato.

```
1 actualFloor = 0 # Si inizializza la variabile da utilizzare
2
3 if floor > 13 :
4     actualFlorr = floor - 1
5 else :
6     actualFloor = floor
```

*Esempio codice 10: esempio ciclo if*

per far sì che il ciclo funzioni e non vada in errore bisogna ricordarsi di rispettare l'indentazione. Il blocco di enunciati è un gruppo di uno o più enunciati incolonnati a sinistra allo stesso modo e posizionati più a destra rispetto all'intestazione. Inoltre, al termine dell'intestazione di un enunciato composto è necessaria la presenza del simbolo dei 'due punti'. Un esempio di sintassi per l'enunciato `if` è riportato nell'esempio 11.

```
1 if condizione :
2     enunciati
3
4 if condizione :
5     enunciati1
6 else :
7     enunciati2
```

*Esempio codice 11: sintassi enunciato if*

Normalmente per verificare se una condizione è vera o falsa si utilizzano degli operatori relazionali, come ==, !=, <, <=, >=.

Spesso è necessario inserire un enunciato `if` all'interno di un altro enunciato `if`: si parla, in tal caso, di enunciati *annidati (nested)*. Il processo decisionale a due livelli si traduce, nel programma che termina il paragrafo, mediante due livelli di enunciati `if`. In teoria il processo di annidamento delle decisioni può coinvolgere anche più di due livelli: un processo di decisione a tre livelli richiede un annidamento a tre livelli. Un esempio di decisioni che comprendono più di due soli casi, che definiamo alternative multiple, è riportato nell'esempio 12. In questo esempio abbiamo il ciclo decisionale che si basa sullo stampare a schermo delle stringhe testuali in base al valore in ingresso. In particolare, si va ad indicare l'entità del danno causato da un terremoto con valore, sulla scala Richter, passato dall'utente.

```
if richter >= 8.0 :
    print ("Most structure fall")
elif richter >= 7.0 :
    print ("Many buildings destroyed")
elif richter >= 6.0 :
    print ("Many buildings considerably damaged, some collapse")
elif richter >= 4.5 :
    print ("Damage to poorly constructed buildings")
else :
    print ("No destruction of buildings")
```

*Esempio codice 12: if ad alternative multiple*

in questo esempio l'enunciato `elif` sta ad indicare la successione di enunciati `else` e `if` in serie.

## 3.7. Cicli

### 3.7.1 Ciclo '*while*'

Uno dei cicli più utilizzati nei linguaggi di programmazione è il ciclo '*while*'. In Python la keyword riservata è `while`. Il ciclo così definito va ad eseguire una serie di enunciati presenti all'interno dell'istruzione `while` che vengono ripetuti finché la condizione è verificata. L'insieme degli enunciati dopo il while costituisce il *corpo (body)* del ciclo. Un esempio di applicazione di un ciclo `while` è riportato nell'esempio 13, in cui si vuole incrementare il contatore degli anni e, conseguentemente, aggiungere interessi finché il saldo è minore dell'obiettivo da raggiungere.

```
while balance < TARGET :
    year = year + 1
    interest = balance * RATE / 100
    balance = balance + interest
```

*Esempio codice 13: ciclo while*

I cicli `while` si suddividono in due tipologie, '*controllato a contatore'* e *'controllato da un evento'*. La differenza alla base risiede nel fatto che i cicli controllati da un evento, come l'esempio 13, vengono eseguiti fino al verificarsi di un determinato evento. Un errore molto ricorrente a cui fare molta attenzione è il *ciclo infinito* che viene eseguito indefinitamente e può essere arrestato soltanto terminando l'esecuzione del programma tramite il sistema operativo oppure riavviando il PC.

Nell'esempio 14 è possibile vedere un esempio di sintassi riservata al ciclo `while`.

```
while condizione :
    enunciati
```

*Esempio codice 14: sintassi ciclo while*

### 3.7.2 Ciclo '*for*'

Molte volte nasce la necessità di voler esaminare tutti i caratteri contenuti in una stringa o tutti i valori contenuti in una lista. Il ciclo `for` può essere usato per eseguire iterativamente istruzioni sugli elementi di qualunque contenitore. Per iterare un determinato enunciato in un intervallo di numeri interi, da Python, è messa a disposizione la funzione `range` che ha proprio lo scopo di generare una sequenza di valori basata sui propri argomenti. La funzione può essere definita andando ad indicare un valore di inizio e fine, in questo caso l'aumento sarà unitario. Se invece si vuole definire un incremento diverso da uno va inserito come terzo argomento della funzione. Nell'esempio 15 possiamo vedere la sintassi dedicata per un ciclo con e senza funzione range e uno spezzone di codice in cui viene definito un range in cui l'incremento è diverso da uno.

```
for variabile in contenitore :  # Sintassi classica ciclo for
    enunciati

for variabile in range (inizio, fine): # Sintassi con utilizzo funz range e incremento unitario
    enunciati

for variabile in range (inizio, fine, incremento): # Sinrassi con utilizzo funz range e incremento da specificare
        enunciati
```

*Esempio codice 15: esempi ciclo for*

## 3.8. Le funzioni

Una funzione (*function)* è una sequenza di istruzioni dotata di un nome che possono essere richiamate in qualsiasi momento. Normalmente si utilizzano quando si vuole rendere il programma più snello, rimandando alcune operazioni ad altre righe di codice, oppure quando c'è la necessità di riutilizzare determinate righe di testo, e per non ripeterle più volte si utilizza una funzione. Per far sì che il main riesca a svolgere le operazioni contenute in una funzione c'è bisogno che essa venga invocata (*called)* al momento del bisogno. Una funzione riceve dei valori in ingresso, che vengono chiamati *argomenti*, e restituisce il risultato dell'elaborazione degli enunciati riportati nella funzione stessa. Per scrivere una funzione bisogna:

- Scegliere un nome per la funzione
- Definire una variabile per ciascuno degli argomenti passati, che prende il nome di *variabili parametro*

Una volta eseguiti questi passaggi si passa alla prima riga della funzione. Necessariamente la prima riga deve iniziare con la parola dedicata `def` seguita dal nome della funzione e dall'argomento passato. Questa riga prende il nome di *intestazione* e la parola `def` fa capire all'interprete che stiamo definendo una funzione. Da qui poi si passa alla definizione del corpo della funzione in cui sono contenuti gli enunciati che producono il risultato. Una regola che in Python va

sempre rispettata è l'indentazione, e nelle funzioni la regola da seguire è la seguente. Gli enunciati vanno posti tutti sulla stessa colonna e distanziati dal margine di un livello, in modo da far capire a Python che quelle righe riguardano la funzione e sono contenute in essa. Per poter collaudare una funzione è necessario definire tutti i parametri definiti come argomenti. Le variabili che hanno il compito di ricevere gli argomenti della funzione sono definite *variabili parametro*, o *parametri fondamentali*. Gli *argomenti*, invece, vengono definiti come *parametri effettivi*. Un'analisi logica su come opera una funzione può essere fatta seguendo dei passaggi molto semplici. Dapprima la variabile parametro viene creata nel momento in cui la funzione viene invocata, la variabile parametro viene inizializzata con il valore dell'argomento passato alla funzione. Da qui la funzione esegue gli enunciati che costituisco il *body* della funzione stessa. La funzione termina e restituisce il controllo del flusso d'esecuzione: tutte le sue variabili vengono eliminate e il valore viene trasmesso all'invocante. Ad una funzione possiamo passare più argomenti andandoli semplicemente ad indicare quando definiamo una funzione separandoli da una virgola. Per ottenere un la restituzione esplicita al chiamante del risultato della funzione dobbiamo utilizzare l'istruzione `return`. Per avere più output basterà semplicemente inserirli nella stessa linea fisica dell'istruzione `return` separandoli da una virgola. Nell'esempio 16 possiamo vedere alcuni esempi di definizione e chiamata di funzioni.

```
def nomeDiFunzione(nomeParametro1, nomeParametro2,.....):  #Sintassi per definizione funzione
    enunciati

def cubeVolume(sideLenght):  #esempio definizione volume con istruzione return
    volume = sideLenght ** 3
    return volume

def operazioni(a, b):  #Esempio funzione in cui immettiamo 2 paratri restituisce 2 output
    somma = a + b
    prodotto = a * b
    return somma, prodotto
```

*Esempio codice 16: esempi funzione*

Per richiamare una funzione basta semplicemente chiamarla ed esplicare, tra le parentesi tonde, gli argomenti della funzione come mostrato nell'esempio 17.

```
result = cubeVolume(2)
# in questo caso assegnamo a result il risultato della funzione cubeVolume a cui passiamo come argomento 2
```

*Esempio codice 17: esempio invocazione funzione*

## 3.9. I moduli

Per avere un programma strutturalmente ordinato o per eseguire dei programmi da linea di comando andiamo ad utilizzare i *moduli*. La libreria standard è struttura in moduli, ciascuno dei quali è un contenitore di attributi. Facendo così riusciamo a creare dei moduli in cui sono risolti dei problemi già noti senza dover ripetere, all'interno del programma, tutte le linee di codice necessarie alla risoluzione del problema stesso. L'esempio più basilare riguarda la libreria `math` che raggruppa al suo interno innumerevoli operazioni di tipo matematico, come ad esempio seno e cose. Per eseguire un modulo bisogna dapprima importarlo nel main utilizzando l'istruzione `import`. Una volta importato per richiamarlo si usa la sintassi indicata nell'esempio 18.

```
def numero(a): # nome di salvataggio= numero.py
    print ("il numero è:",a) # il modulo semplicemente stampa a schermo il numero passato

import numero # importo il modulo
a = 3 #definisco l'argomento
nunero.numero(a) #richiamo il modulo

nome_file.nome_modulo(argomento)
# la regola generale mi dice che per richiamare un modulo devo prima mettere il nome di salvataggio del modulo
#seguito dal nome del modulo, il testo che segue la parola def, e poi tra parantesi l'argomento che gli passo
```

*Esempio codice 18: chiamata modulo*

## 3.10. Principali differenze tra Matlab e Python

Una delle principali differenze tra Matlab e Python, tralasciando gli aspetti commerciali delle licenze, può essere identificata nel linguaggio stesso. Per definizione, Python è un linguaggio di programmazione multiparadigm, generico, scritto in ANSI C portatile e utilizzato per sviluppare applicazioni a pieno titolo o altri strumenti software. Il linguaggio e le librerie di base di Python funzionano allo stesso modo su tutte le piattaforme, quindi si può programmare in ambienti misti. Matlab, d'altra parte, è un linguaggio commerciale matematico orientato alla matrice per la programmazione matematica. Matlab ha toolbox specializzati che sono sviluppati professionalmente e meticolosamente testati per diverse applicazioni scientifiche e ingegneristiche. Matlab ha funzionalità numeriche più complete di Python.
L'altra differenza riguarda le librerie messe a disposizioni da entrambi gli sviluppatori. In Python, essendo open source e di carattere generale, possiamo trovare svariate librerie basate su import/export di file per networking o comunque librerie di carattere più generale rispetto a Matlab in cui le librerie sono solo di carattere matematico.
Una delle differenze alla quale, secondo me, è da prestare molta attenzione riguarda l'indicizzazione. In Python, come in Java e C++, l'indice parte da 0 e non da 1 come in Matlab. Quindi quando si fa la conversione di script da un linguaggio all'altro bisogna ricordarsi di questa differenza e aggiornare i dati o gli script stessi.

## 3.11. Librerie

### 3.11.1 Libreria math

Il modulo math fornisce l'accesso alle funzioni matematiche definite dallo standard C. Questa libreria funziona solo se l'argomento che gli viene passato non appartiene all'insieme dei numeri complessi. La libreria math consente di andare ad utilizzare le principali funzioni matematiche, come seno, coseno, logaritmi, che in genere in Python non sono comprese.

### 3.11.2 Libreria NumPy

Il modulo NumPy, nato nel 2005 da Travis Oliphant, è una delle librerie più importanti per il passaggio da script Matlab a Python in quanto aggiunge un notevole supporto ad operazione con matrici e array. Come quasi tutte le librerie disponibile su Python, anche NumPy è open source e continuamente aggiornata. La libreria è disponibile per il download sul sito NumPy.org. Oltre al download nel sito possiamo trovare tutta la documentazione inerente alla funzione, guide all'istallazione e guide a tutte le funzioni definite nel modulo stesso.

### 3.11.3 Libreria SciPy

Anche la libreria SciPy come la NumPy è stata realizzata da Travis Oliphant nel 1999. Il modulo consente di lavorare con degli algoritmi e strumenti matematici da implementare a Python. Contiene moduli per l'ottimizzazione, per l'algebra lineare, l'integrazione e molti altri strumenti comuni nella scienza e nell'ingegneria. Ovviamente come tutte le librerie che girano intorno a Python, anche SciPy è open source e anche in questo caso troveremo un sito dedicato, scpy.org, in cui trovare qualsiasi tipo di informazione.

# 4

# DESCRIZIONE MODULI

Dopo un ampio richiamo alle grandezze fisiche che sono calcolate nei moduli e dopo una presentazione del linguaggio di programmazione Python e le librerie che mi hanno fatto da ponte tra il linguaggio di programmazione da Matlab a Python, introduco i moduli.

## 4.1. Displacement

### 4.1.1 Presentazione dello script

La funzione Displacement va a calcolare il volume in funzione dell'angolo di manovella. Di seguito nella figura 11 è mostrato l'intero codice di cui andremo ad analizzare gli input/output e le linee logiche più importanti.

```
1  import math
2  import numpy as np
3  import scipy.linalg
4  def Displacement(d, r, l, pinoff, rc, Res):
5      d = d / 1000
6      r = r / 1000
7      l = l / 1000
8      pinoff = pinoff / 1000
9
10     _lambda = r / l
11     A = math.pi * d ** 2 / 4
12
13     angolo = math.sqrt(1 - (pinoff / (l + r)) ** 2)
14     angolo_rad = math.radians(angolo)
15     RefOff = math.cos(angolo_rad)
16     if pinoff < 0:
17         RefOff = -RefOff
18
19
20
21     N = int(720 / Res)
22     theta = np.linspace(-360, 360, N) + RefOff
23
24
25
26     theta_rad = np.radians(theta)
27     V = (A * 2 * r)/(rc-1)+A * l *((1+_lambda) * np.sqrt(1-pinoff/(l+r)**2)-_lambda * np.cos(theta_rad)-np.sqrt(1-
   ((r * np.sin(theta_rad)-pinoff)/1)**2))
28     return V, theta
```

*Figura 11: codice completo Displacement*

### 4.1.2 Analisi input/output

**INPUT:**

- d è uno scalare e indica l'alesaggio
- r è uno scalare e indica il raggio di manovella
- l è uno scalare e indica la lunghezza di biella
- pinoff è uno scalare e indica l'eventuale offset dell'asse dello spinotto rispetto all'asse del pistone
- rc è anch'esso uno scalare e indica il rapporto di compressione
- Res invece indica la risoluzione angolare che si vuole adottare per la generazione della matrice dell'angolo di manovella

**OUTPUT:**

- V indica il volume. È un vettore in quanto dipendente dall'angolo di manovella
- theta è un vettore e rappresenta la corrispondente posizione angolare

### 4.1.3 Studio dettagliato codice

```
1 import math
2 import numpy as np
3 import scipy.linalg
```

*Displacement 1*

Nelle prime tre righe di codice vengono richiamate le librerie che poi saranno utilizzate nel corpo del modulo usando l'istruzione `import` già definita nel capitolo 3.
Una volta importate le librerie, nella riga 4 viene definito il modulo Displacement.
Dalla riga 5 alla riga 8 invece troviamo le conversioni degli input, che essendo forniti in millimetri, devono essere convertiti in metri.
Nella linea 10 e 11 troviamo la definizione di lambda, come rapporto tra il raggio di manovella e la lunghezza di biella, e la sezione trasversale. Nel calcolo della sezione trasversale troviamo la prima applicazione del modulo `math` per il calcolo del pigreco.
Le linee 13 e 14, riguardano il calcolo dell'angolo di offset. In particolare, la linea 14 è utile perché il modulo `math.cos` (richiamato nella linea 15) accetta come input l'angolo in radianti.
Nelle linee 15-17 troviamo la definizione di RefOff e successivamente un'istruzione di tipo decisionale `if` che non fa nient'altro che cambiare segno a RefOff nel caso in cui pinoff è negativo.
Le successive linee, 21 e 22, invece generano il vettore angolare theta. In particolare, nella line 21 si va a definire il passo che poi servirà per la funzione `linspace`.
Infatti `np.linspace` riceve come argomenti (`start, end, step`) e avremo come risultato un vettore che parte da -360°, finisce a 360° con un passo di N.

Una volta definito il vettore theta si va ad applicare la funzione `np.radians` per convertire il vettore e, usando la 1.5 , si va a calcolare il volume.
L'ultima riga ritorna al chiamante i valori di V e theta.

## 4.2. Imepshl

### 4.2.1 Presentazione dello script

Questo modulo è utilizzato per il calcolo della IMEP in diverse condizioni.

```
1  import numpy as np
2  def imepshl(P, V, win_st, win_end):
3      Vc=max(V)-min(V)
4      IMEP = (np.trapz(P*10**5,V))/(Vc)/(10**5)
5      V1 = V[win_st:win_end]
6      P1 = (P[win_st:win_end,:])*(10**5)
7      IMEPH = (np.trapz(P1,V1))/(Vc)/(10**5)
8      IMEPL=IMEP-IMEPH
9      return IMEP, IMEPH, IMEPL
```

*Figura 12: codice completo imepshl*

### 4.2.2 Analisi input/output

**INPUT:**

- P: è una matrice e rappresenta la pressione misurata in camera
- V: è un vettore e rappresenta il volume
- win_st: è uno scalare ed è il valore che indica l'inizio della finestra che ci interessa verificare.
- win_end: è uno scalare ed è il valore che indica la fine della finestra che ci interessa verificare

IMPORTANTE: in relazione alle finestre, bisogna ricordarsi che se prendiamo i dati importati da Matlab bisogna andare a sottrarre 1 al valore di win_st in quanto Python inizia l'indicizzazione da 0

**OUTPUT:**

- IMEP: è la pressione media indicata, è un vettore e la formula utilizzata per calcolarla è la 2.2.9
- IMEPH: è un vettore che sta ad indicare la IMEP calcolata a valvole chiuse. La formula utilizzata per il calcolo è la 2.2.10. È la IMEP che si otterrebbe solo nella fase in cui le valvole sono chiuse ed è un valore maggiore della IMEP in quanto la IMEPH non tiene conto della fase di pompaggio.
- IMEPL: è anch'esso un vettore ed è la IMEP calcolata a valvole aperte e la formula utilizzata per calcolarla è la 2.2.11. Dalla IMEPL ci si dovrebbe

aspettare un valore negativo in quanto è calcolata con le valvole aperte nella fase di pompaggio. Quindi dovrebbe essere quella che ci mette in evidenza la presenza del ciclo di pompaggio acquisendo dei valori molto negativi quando il ciclo di pompaggio è evidente e prossimi allo zero quando il ciclo non è più rilevante.

### 4.2.3 Studio dettagliato del codice

Come tutti i moduli che tratterò in questo capitolo, le prime righe sono sempre riservate all'importazioni di moduli che poi saranno utili allo svolgimento del programma.
Nella linea 2 abbiamo l'intestazione del modulo, in cui andiamo a richiamare il nome del modulo e tutti gli argomenti passati come input.
Di seguito nella linea 3 abbiamo la definizione di Vc come differenza tra il valore massimo e il valore minimo del vettore V.
Nella 4 abbiamo invece il calcolo della IMEP utilizzando la 2.2.9. La regola di integrazione è la trapezoidale che prende come input due valori che sono la pressione in pascal e la cilindrata. In particolare, va a calcolare l'integrale della P rispetto a V. Il risultato viene poi diviso per Vc e si ritorna in bar.
Le linee 5 e 6 servono per creare delle variabili di appoggio che saranno poi utilizzate per il calcolo della IMEPH. La formula utilizzata è la 2.2.10 mentre la regola per l'integrazione è sempre la regola del trapezio. Ovviamente nel calcolo delle IMEP non la si vuole andare a calcolarle su tutti i valori di P e di V e per questo si vanno ad individuare dei vettori/matrici che sono il risultato dello slicing, usando i valori di win_st e win_end, di P e di V. Questo è visibile sia nel calcolo della IMEP, linea 4, che nel calcolo della IMEPH, quando si creano le variabili di appoggio V1 e P1 nelle linee 5 e 6.
L'ultima riga è caratterizzata dall'istruzione `return` che mi restituisce i valori degli output. Una cosa molto importante è andarsi a ricordare l'ordine degli argomenti indicati in questa riga, perché sarà poi l'ordine dei valori restituiti all'invocatore.

## 4.3 Knock

### 4.3.1 Presentazione dello script

La funzione knock ha il compito di andare a calcolare l'indice di detonazione seguendo due metodi, il MAPO o l'INT. Di seguito verrà visualizzato e analizzato in dettaglio l'intero codice.

```
1  import numpy as np
2  import scipy.linalg
3
4  def Knock(P_hi, win_st, win_end, method):
5      N = P_hi[win_st:win_end,:]
6      __switch_0__ = method
7      if 0:
8          pass
9      elif __switch_0__== 'MAPO':
10         KI = np.ndarray.max(abs(N))
11
12     elif __switch_0__ == 'INT':
13         KI = np.sum(abs(N))
14     return KI
```

*Figura 13: codice completo knock*

### 4.3.2 Analisi input/output

**INPUT:**

- P_hi: è un vettore e contiene il ciclo di pressione filtrato passa alto
- win_st: è uno scalare e indica l'inizio della finestra su cui vogliamo calcolare l'indice di detonazione
- win_end: è anch'esso uno scalare e indica la fine della finestra su cui vogliamo calcolare l'indice di detonazione
- method: è una stringa che mi sta ad indicare il metodo con il quale voglio sia calcolato il KI

**OUTPUT**

- KI: è o uno scalare o un vettore, dipende dalla tipologia dell'ingresso P_hi e indica l'indice di detonazione.

### 4.3.3 Studio dettagliato del codice

Le prime righe richiamano le due librerie che ho utilizzato di più per la realizzazione di questi codici, che sono la NumPy e la SciPy, descritte nel paragrafo 3.11.
La riga 4 contiene l'intestazione del modulo.
Nella 5 invece andiamo a compiere uno slicing del vettore P_hi andando a prendere solo i valori che rientrano nella finestra sulla quale vogliamo porre attenzione.
Nella riga 6 invece abbiamo la definizione della variabile switch che viene eguagliata alla stringa di testo che viene passata come argomento della funzione.
Una volta definita la variabile switch, andiamo a richiamare l'enunciato decisionale `if` spiegandogli che, se method è uguale a 0, allora l'istruzione successiva sarà quella di 'passare' l'istruzione stessa, se invece switch è uguale a MAPO, allora si calcola KI utilizzando la 2.2.21, se invece è uguale a INT allora si utilizza la 2.2.22.
L'ultima riga restituisce il valore KI

## 4.4. Peak rise rate

### 4.4.1. Presentazione dello script

Il modulo va a calcolare il tasso di aumento del picco della pressione. Nella figura 14 il codice completo.

```python
import numpy as np
import scipy.linalg

def PeakRiseRate(P, win_st, win_end, theta):
    idx_ROHRStart = np.flatnonzero(theta>=win_st)[2]
    idx_ROHREnd = np.flatnonzero(theta>=win_end)[2]

    nCycle = P.shape[1]

    dP = np.empty(P.shape)
    dP = np.NaN
    for ii_cyc in range (0,nCycle):
        dP[:,ii_cyc] = np.gradient(P[:,ii_cyc])

    theta_srch = theta[idx_ROHRStart:idx_ROHREnd]
    dP_srch = dP[idx_ROHRStart:idx_ROHREnd,:]

    dPmax = np.amax(dP_srch)
    imax = np.argmax(dPmax)
    A_dPmax = theta_srch[imax]
    return dPmax, A_dPmax, dP
```

*Figura 14: codice completo PeakRiseRate*

### 4.4.2 Analisi input/output

**INPUT:**

- P: è una matrice che rappresenta la pressione
- win_st: è uno scalare che rappresenta l'inizio della finestra di interesse
- win_end: è uno scalare che rappresenta la fine della finestra di interesse
- theta: è il vettore che indica l'angolo di manovella

**OUTPUT:**

- dPmax: è un vettore ed è la massima variazione di pressione in termini di bar per grado
- A_dPmax: è un vettore ed è la corrispondente angolare della massima variazione di pressione
- dP: è una matrice e fa vedere l'andamento della derivata della pressione

### 4.4.3 Studio dettagliato del codice

Una volta richiamate le funzioni da utilizzare e definito il modulo, si vanno a definire due variabili che corrispondono agli indici del ROHR start e del ROHR end.
Le linee 5 e 6 assegnano alle variabili ROHR start e ROHR end il primo valore diverso da 0 maggiore di win_st e win_end nel vettore theta.
La riga 8, con la funzione `shape`, assegna alla variabile `nCycle` un vettore riga di lunghezza uguale a P.
Da qui nella riga 10 e 11 dP va a creare un vettore di NaN di dimensione uguale a P.
Le linee da 12 e 13 vanno a calcolare la dP andando a calcolare la derivata elemento per elemento e riempiendo la matrice creata in precedenza con i NaN.
Le variabili create nelle linee 15 e 16 vanno a compiere uno slicing del vettore theta, riferito agli indici calcolati nelle righe 5 e 6. Stessa cosa avviene per la matrice dP.
Le linee 18 e 19 vanno a cercare il massimo del vettore dP_srch e il rispettivo indice imax.
Nella linea 20 viene assegnato ad A_dPmax il corrispondente valore di theta_srch corrispondente all'indice imax.

## 4.5. Pegging

### 4.5.1 Presentazione dello script

Il segnale proveniente dal sensore piezoelettrico e passante attraverso l'amplificatore di carica è un segnale privo della sua componente media, cioè filtrato passa alto con una frequenza di taglio molto bassa. La funzione pegging ricostruisce la componente media tagliata dal filtro passa alto.

```
1  import numpy as np
2  import scipy.linalg
3
4  def pegging(P, V, win_st, win_end, win_dur, poly_coef, ref_value, win_ref, method):
5      __switch_0__ = method
6      if 0:
7          pass
8      elif __switch_0__ == 'poly' :
9
10         win_st_minus = win_st-win_dur/2
11         win_st_plus = win_st+win_dur/2
12         V1 = V[win_st_minus:win_st_plus]
13         P1 = P[win_st_minus:win_st_plus,:]
14         win_end_minus = win_end-win_dur/2
15         win_end_plus = win_end+win_dur/2
16         V2 = V[win_end_minus:win_end_plus]
17         P2 = P[win_end_minus:win_end_plus,:]
18
19         Peg_mean = ((P2*(V2.T*np.ones(P2.shape[1])**poly_coef)-P1*
   (V1.T*np.ones(np.shape[1]))**poly_coef)/((V1.T*np.ones(P2.shape[1]))**poly_coef)-(V2.T*np.ones(P2.shape[1]))**poly_coef)
20         Peg = Peg_mean.mean(axis=0)
21     elif __switch_0__ == 'ref':
22         win_ref_minus = win_ref-win_dur/2
23         win_ref_plus = win_ref+win_dur/2
24         Peg = ref_value-(P[win_ref_minus:win_ref_plus,:])
25     Ppeg = P + np.ones(P.shape[0]) * Peg
26     return Ppeg
```

*Figura 15: codice completo pegging*

### 4.5.2 Analisi input/output

**INPUT:**

- P: è la matrice della pressione
- V: è un vettore e rappresenta il volume
- win_st: è uno scalare e rappresenta l'inizio della finestra che ci interessa esaminare
- win_end: è uno scalare e rappresenta la fine della finestra che ci interessa esaminare
- win_dur: è uno scalare e rappresenta la durata della finestra
- poly_coeff: è uno scalare ed è il coefficiente della politropica del tipo p$V^n$=cost
- ref_value: è un vettore ed è il valore di riferimento, la natura, quindi scalare o vettore, dipende dalla natura di P, se P è vettore allora ref_value è uno scalare, se P è una matrice ref_value è un vettore
- method: stinga di testo che indica il metodo da utilizzare

**OUTPUT:**

- Ppeg: matrice che corrisponde alla componente media

### 4.5.3 Studio dettagliato del codice

Dopo l'intestazione della funzione viene inizializzata la variabile switch ed eguagliata alla stringa di testo passata come input.
Da qui si inizializza un enunciato decisionale in cui si va ad indicare il metodo da utilizzare se POLY o REF.
Una volta definito il metodo, se è uguale a POLY, allora il programma continua la lettura dalla linea 10, in cui sono definite delle variabili di appoggio che poi saranno necessarie per lo slicing del vettore V e della matrice P, che avviene nelle linee 12 e 13.
La stessa cosa è poi ripetuta nelle linee 14 e 15, in cui si definiscono altre due variabili che poi saranno utilizzate nel calcolo della Peg_mean.
Nella linea 19 si va a creare una variabile, anch'essa d'appoggio, per snellire la sintassi del programma perché, per andare ad applicare la funzione `mean` ad un array, è necessario dichiararlo prima della funzione stessa in questo modo `matrice.mean`.
Se invece il method è uguale a REF allora l'interprete salta tutto il blocco che va dalla riga 10 alla riga 20 per eseguire le istruzioni contenute nella linea 22 in cui, anche qui, vengono definite due variabili di appoggio utili per lo sclicing della linea 24.

## 4.6 Taglio campane

### 4.6.1 Presentazione dello script

Il modulo taglio campane va a restituire come output una matrice di punti, che se plottati formano le campane di pressione

```python
import numpy as np
def taglioCampane(Ang, Pres, Abs):
    #Funzione per il taglio delle campane
    #Togliere i NaN
    Abs = Abs[~np.isnan(Abs)]
    Ang = Ang[~np.isnan(Ang)]
    Pres = Pres[~np.isnan(Ang)]
    start = np.nonzero(np.diff(Ang)<0)[0][1] + 1
    ending = np.nonzero(np.diff(Ang)<0)[0][1]
    Ang = Ang[start-1:ending]
    Pres = Pres[start-1:ending]
    index = np.nonzero(np.diff(Ang))[0][1]
    xq = np.arange(-360,360,0.1)

    C = np.empty((len(xq),len(index),))
    C[:] = np.nan
    #Nel range del for si inzia a contare da 0 perchè Python
    #inizia a contare da 0, diversamente da Matlab che inizia a contare da 1
    kk = 0
    for ii in range(0,len(index)):
        a = 1+kk
        b = index[ii]

        #sostituzione funzione unique 'stable'
        theta, idx, ic = np.unique(Ang[a-1:b], return_index=True, return_reverse=True)
        #Bisogna invertire theta
        theta = theta[np.argsort(idx)]
        #bisogna invertire idx, aggiungo +1 perchè gli indici iniziano da 0
        idx = idx[::-1]+1

        Pressure = Pres[a-1:b]

        Pressure_test = Pressure(idx)
        C[:,ii] = np.interp(theta,xq,Pressure_test)
        kk=index[ii]
```

*Figura 16: codice completo taglio campane*

### 4.6.2 Analisi input/output

**INPUT:**

- Ang: è vettore ed è una corrispondenza angolare
- Pres: è la matrice della pressione
- Abs: è vettore ed è la posizione angolare assoluta

**OUTPUT:**

- C: è la matrice di punti che se plottati restituisce la campana

### 4.6.3 Studio dettagliato del codice

Dopo aver importato, nella riga 1, la libreria NumPy e aver definito l'intestazione della funzione, nelle righe che vanno dalla 5 alla 7, l'interprete va ad eliminare gli eventuali NaN contenuti negli input.
Nelle righe 8 e 9, invece, si identificano i punti di partenze, start, e arrivo, end, dello slicing delle variabili Ang e Pres, che poi avviene nelle linee 10 e 11.
La variabile xq, indicata nella linea 13, invece sarà un vettore che spazia da -360° a 360° con un passo di 0.1.
Nella linea 15 invece viene inizializzata la matrice C, che avrà le colonne uguali alla lunghezza di index e le righe uguali alla lunghezza di xq. Alla matrice creata gli assegniamo tutti valori NaN.
Inizializziamo una variabile, kk, che poi servirà da contatore nel cilco for, definito nella linea 20, che avrà un range di applicazione che andrà da 0 alla lunghezza di index.
Da qui particolare importanza va riservata alla line 25. La funzione `unique` che va a trovare gli elementi ripetuti e ritorna tre output, in cui il primo è la variabile a cui ha tolto le doppie mentre gli altri due output riguardano gli indici e il numero di volte che compare un elemento ripetuto nell'array di partenza.
Da qui la riga più importante a cui fare riferimento è la 34. In questa linea, alla matrice già creata in precedenza C, gli viene assegnato il risultato di un'interpolazione lineare a 1 dimensione dei vettori theta e Pressure_test, dove Pressure_test è una funzione di theta e l'interpolazione è fatta seguendo i punti richiesti presi dal vettore xq.

# Conclusione

Il linguaggio di programmazione Python, essendo un linguaggio molto versatile, è uno dei più utilizzati per svariate applicazioni di carattere scientifico e non scientifico. Il suo grande pregio risiede anche nel fatto che, essendo open source, diventa un ottimo strumento multipiattaforma sempre aggiornato in cui, grazie alle svariate librerie rilasciate gratuitamente da informatici professionisti, è possibile la scrittura di programmi di varia natura. Oltretutto la sua struttura e la sua relativa velocità di compilazione lo rende adatto a calcoli di moli di dati importanti come il nostro caso, in cui i dati relativi a prove al banco costituiscono dei pacchetti abbastanza ampi da analizzare.
D'altro canto, come dicevo prima, essendo un linguaggio non studiato direttamente per applicazioni di tipo scientifico, Python ha bisogno di integrazioni per realizzare calcoli di carattere matematico. Per questo ci sono le librerie dedicate atte allo scopo di integrare al linguaggio operazioni di carattere puramente matematico di cui il linguaggio è sprovvisto. A mio avviso, però, l'utilizzo di librerie di terzi e non sviluppate direttamente da chi gestisce l'evoluzione del programma potrebbe portare ad errori sia sintattici e sia errori che riguardano il risultato dell'operazione richiesta. Oltre a questo, in alcuni casi, le librerie installate vanno incontro ad incompatibilità con determinati aggiornamenti di alcuni sistemi operativi rendendo impossibile la compilazione di script che contengono le suddette librerie.
Un altro appunto necessario è da fare sul fatto che la scrittura in Python per un programmatore che passa da un sistema di programmazione orientato alla matrice, come Matlab, non è di immediata realizzazione. Questo perché, ad esempio, se si proviene da Matlab, bisogna ricordarsi la differenza riguardante l'indicizzazione che in Python inizia da 0 e in Matlab da 1. La conseguenza prodotta da questa differenza risiede nella necessità di dover modificare i dati in ingresso, o lo script, per avere dei risultati corretti. Se invece il passaggio avviene da linguaggi di tipo C++ o Java, il problema dell'indicizzazione non si pone in quanto anche essi partono da 0.

# Bibliografia

[1] G. Ferrari, Motori a combustione interna, Società Editrice Esculapio, 2016

[2] G. Cantore, Fondamenti di motori endotermici alternativi, Società Editrice Esculapio, 2008

[3] E. Corti, Appunti del corso di laboratorio di motori a combustione interna, A.A. 2017-2018

[4] M. Buttu, Python guida completa, Edizioni LSWR, 2014

[5] C. Horstmann R. D. Necaise, Concetti fondamentali di informatica e fondamenti di Python, Maggioli Editore, 2019

# Sitografia

[1] Sito ufficiale Python: www.python.org

[2] Sito ufficiale libreria NumPy: numpy.org

[3] Sito ufficiale libreria SciPy: scipy.org

# Ringraziamenti

Prima di tutto mi sento di ringraziare il professor Enrico Corti che si è mostrato sempre disponibile nei miei confronti e a cui aspiro a livello professionale.
Un ringraziamento speciale va alla mia famiglia che mi ha insegnato a camminare sempre a testa alta affrontando con razionalità e fermezza tutti i problemi che la vita può riservarmi. Mi sento anche di ringraziare tutti gli zii, le zie e i cugini che hanno reso possibile che questo mio sogno diventi realtà. Ed infine vorrei ringraziare tutte quelle persone che ho incontrato in questo percorso che, in maniera diretta o indiretta, mi hanno aiutato e supportato.