Analisi Statistica dei Dati Sperimentali

Laboratorio di Fisica

1 Introduzione

In questo capitolo esploreremo l'analisi statistica dei dati sperimentali utilizzando Python. Gli argomenti trattati includeranno la gestione delle misure come variabili casuali, la funzione di distribuzione normale, l'uso di matplotlib per la visualizzazione dei dati e la regressione lineare. Eseguiremo anche un esempio completo per il calcolo dell'accelerazione di gravità.

2 Misure come Variabili Casuali

Le misure sperimentali sono trattate come variabili casuali, il che implica che i valori misurati sono soggetti a variazioni casuali dovute a errori di misurazione e condizioni sperimentali. Tutto questo, nell'ipotesi che la sensibilità degli strumenti non copra le fluttuazioni casuali. Se misuro la lunghezza di un tavolo 40 volte, otterrò sempre lo stesso valore, se uso un metro da falegname. In tal caso la statistica è inutile perché i dati non mostrano alcuna variabilità. Se però misuro il tempo di caduta di una pallina che viene lasciata cadere da una persona e lo faccio con un cronometro manule, otterrò diversi valori e in tal caso ha senso fare un'analisi statistica. Nelle pagine che seguono, supporremo di fare misure che esibiscono una certa variabilità. Un esempio è la misura delle altezze di una popolazione di persone (ad esempio uomini della stessa età). Sappiamo che otterremo una certa variazione ma in tal caso la variabilità non è legata alla persona che esegue le misure ma alla grandezza fisica stessa: le altezze non sono tutte uguali! La statistica, però, non fà distinzione tra questi due casi e si occupa di capire due cose: 1) Qual è il valore medio della grandezza; 2) Come varia questa proprietà. 100 uomini di un gruppo A, potrebbero avere un'altezza media di 1,75 m e cosi anche 100 uomini di un gruppo B ma con una grande variabilità (ad esempio 5 uomini sotto i 160 cm, 5 uominitra 160 cm e 165 cm, 5 uomini tra 165 cm e 170 cm e così via. Nel gruppo A invece, magari i 3/4 degli uomini hanno un'altezza molto vicina alla media. In fisica, lo "sparpagliamento"

dei risultati, viene identificato con quello che abbiamo chiamato errore assoluto o incertezza in quanto ci dà una misura di quanto è precisa la nostra misurazione. Se faccio oscillare un pendolo e misuro i periodi (la durata di un'oscillazione) e il cronometro ha una sensibilità di 0,01 s, ma in media i miei valori si discostano dal valore medio per 0,5 secondi, allora prenderò questa come misura di incertezza.

3 Uso di Matplotlib per la Visualizzazione dei Dati

Matplotlib è una libreria potente per la creazione di grafici in Python. Di seguito è riportato un esempio di come realizzare un grafico delle misure sperimentali con barre di errore.

3.1 Esempio di Grafico

Consideriamo i dati nella tabella 1:

Variabile indipendente (x)	**Variabile dipendente** (y)
0	0.5
1.11	3.1
2.22	5.2
3.33	8.3
4.44	11.1
5.55	13.9
6.66	16.5
7.77	19.4
8.88	22.1
10	25.3

Tabella 1: Dati di x e y utilizzati nel grafico

Il codice Python per realizzare il grafico è:

```
import matplotlib.pyplot as plt
import numpy as np

# Dati di esempio assegnati esplicitamente
x = np.array([0, 1.11, 2.22, 3.33, 4.44, 5.55, 6.66, 7.77,
   8.88, 10])
y = np.array([0.5, 3.1, 5.2, 8.3, 11.1, 13.9, 16.5, 19.4,
   22.1, 25.3])
```

```
yerr = np.array([0.6, 0.4, 0.5, 0.3, 0.7, 0.2, 0.4, 0.6, 0.5,
    0.3])  # Assicurarsi che yerr sia positivo

# Creazione della figura e dell'asse
fig, ax = plt.subplots()

#

ax.errorbar(x, y, yerr=yerr, fmt='o', ecolor='red', capsize
   =5, label='Misure sperimentali')

# Aggiunta di etichette agli assi
ax.set_xlabel('Variabile indipendente')
ax.set_ylabel('Variabile dipendente')

# Aggiunta di un titolo
ax.set_title('Grafico delle Misure Sperimentali con Barre di
   Errore')

# Aggiunta di una legenda
ax.legend()

# Aggiunta di una griglia
ax.grid(True)
# Stampa il grafico
plt.savefig('grafico-misure.png')
# Mostra il grafico
plt.show()
#se vuoi scaricare il file da google colab decommenta le
   righe:
#from google.colab import files
#files.download('grafico-misure.png')
```

Listing 1: Grafico delle misure sperimentali con barre di errore

4 Spiegazione del Codice per un Grafico Sperimentale

4.1 Importazione delle librerie

```
import matplotlib.pyplot as plt
import numpy as np
```

Listing 2: Importazione delle librerie

Il codice sopra importa le librerie necessarie per la creazione del grafico. La libreria `matplotlib.pyplot` fornisce strumenti per la creazione di grafici e visualizzazioni, mentre `numpy` è fondamentale per operazioni numeriche e manipolazione di array.

4.2 Dati di esempio

```
x = np.array([0, 1.11, 2.22, 3.33, 4.44, 5.55, 6.66, 7.77,
    8.88, 10])
y = np.array([0.5, 3.1, 5.2, 8.3, 11.1, 13.9, 16.5, 19.4,
    22.1, 25.3])
yerr = np.array([0.6, 0.4, 0.5, 0.3, 0.7, 0.2, 0.4, 0.6, 0.5,
     0.3])
```

Listing 3: Dati di esempio

Questi array contengono i dati di esempio per il grafico. L'array `x` rappresenta i valori della variabile indipendente, mentre `y` rappresenta i valori della variabile dipendente. `yerr` contiene gli errori associati ai valori di `y`.

4.3 Creazione della figura e dell'asse

```
fig, ax = plt.subplots()
```

Listing 4: Creazione della figura e dell'asse

Questa riga di codice crea una nuova figura e un asse (grafico). La variabile `fig` rappresenta la figura, mentre `ax` è l'asse su cui viene tracciato il grafico.

4.4 Creazione del grafico con barre di errore

```
ax.errorbar(x, y, yerr=yerr, fmt='o', ecolor='red', capsize
    =5, label='Misure␣sperimentali')
```

Listing 5: Creazione del grafico con barre di errore

Questo comando crea un grafico a dispersione con barre di errore. I parametri specificati sono:

- `x`: Array dei valori dell'asse x.
- `y`: Array dei valori dell'asse y.
- `yerr`: Array degli errori associati ai valori di `y`.

- `fmt='o'`: Specifica che i punti dati devono essere rappresentati come cerchi.
- `ecolor='red'`: Colore delle barre di errore (rosso in questo caso).
- `capsize=5`: Dimensione delle estremità delle barre di errore.
- `label='Misure sperimentali'`: Etichetta da utilizzare nella legenda per il grafico a dispersione.

4.5 Aggiunta di etichette agli assi

```
ax.set_xlabel('Variabile␣indipendente')
ax.set_ylabel('Variabile␣dipendente')
```

Listing 6: Aggiunta di etichette agli assi

Queste righe di codice aggiungono etichette agli assi del grafico. `ax.set_xlabel` aggiunge un'etichetta all'asse delle ascisse (x), mentre `ax.set_ylabel` aggiunge un'etichetta all'asse delle ordinate (y).

4.6 Aggiunta di un titolo

```
ax.set_title('Grafico␣delle␣Misure␣Sperimentali␣con␣Barre␣di␣
    Errore')
```

Listing 7: Aggiunta di un titolo

Questa riga di codice aggiunge un titolo al grafico.

4.7 Aggiunta di una legenda

```
ax.legend()
```

Listing 8: Aggiunta di una legenda

Il comando `ax.legend()` aggiunge una legenda al grafico, mostrando le etichette dei dati.

4.8 Aggiunta di una griglia

```
ax.grid(True)
```

Listing 9: Aggiunta di una griglia

Questa riga di codice aggiunge una griglia al grafico, facilitando la lettura dei valori.

4.9 Visualizzazione del grafico

```
plt.show()
```

Listing 10: Visualizzazione del grafico

Infine, `plt.show()` visualizza il grafico creato. Senza questo comando, il grafico non verrà visualizzato.

5 Istogrammi a Bins

Un istogramma è una rappresentazione grafica della distribuzione di un insieme di dati. Esso suddivide l'intervallo dei dati in una serie di intervalli chiamati bins (o classi) e conta il numero di dati che rientrano in ciascun bin. L'altezza di un rettangolo è pari alla frequenza, ossia è pari al rapporto tra le misure comprese tra i due estremi della base del rettangolo e il totale delle misure. In figura 1 vediamo un tipico istogramma.

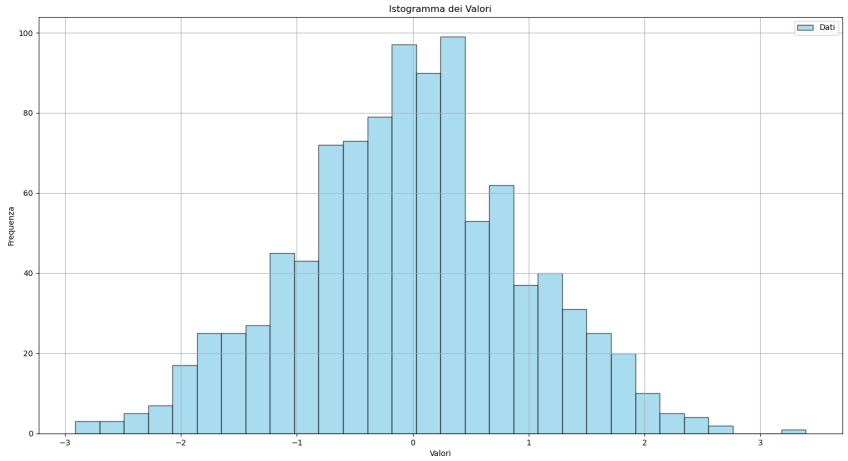

Figura 1: Istogramma dei Valori

L'ampiezza dei bins può influenzare significativamente l'aspetto dell'istogramma. Bins più stretti possono mostrare più dettagli, ma possono anche evidenziare il rumore nei dati, mentre bins più larghi forniscono una visione più smussata della distribuzione. La scelta del numero di suddivisioni dipende dal numero di dati: se si hanno molti dati si possono prendere più bins. Si consiglia di fare dei tentativi. Se l'istogramma rappresenta una distribuzione gaussiana (detta anche "normale") il suo aspetto deve ricordare quello di una

campana. Ne parleremo nei prossimi paragrafi in cui useremo il linguaggio Python per disegnare un'istogramma, dunque non ci dovremo preoccupare di disegnarlo a mano ma è importante conoscere il significato di questo tipo di diagramma. Questo linguaggio offre maggiori personalizzazioni dei semplici grafici realizzati da excel. Il grafico in figura è stato realizzato col codice python:

```
import matplotlib.pyplot as plt
import numpy as np

# Dati di esempio
data = np.random.randn(1000)

# Creazione della figura e dell'asse
fig, ax = plt.subplots()

# Creazione dell'istogramma
n, bins, patches = ax.hist(data, bins=30, color='skyblue',
    edgecolor='black', alpha=0.7)

# Aggiunta di etichette agli assi
ax.set_xlabel('Valori')
ax.set_ylabel('Frequenza')

# Aggiunta di un titolo
ax.set_title('Istogramma dei Valori')

# Aggiunta di una legenda
ax.legend(['Dati'])

# Aggiunta di una griglia
ax.grid(True)
plt.savefig('istogramma.png')
#se vuoi scaricare il file da google colab decommenta le
    righe:
#from google.colab import files
#files.download('istogramma.png')
# Mostra il grafico
plt.show()
```

Listing 11: Script Python per disegnare un'istogramma

6 Spiegazione del Codice per creare un'istogramma

6.1 Importazione delle Librerie

Il codice inizia importando le librerie necessarie:

```
import matplotlib.pyplot as plt
import numpy as np
```

`matplotlib.pyplot` è utilizzata per creare grafici, mentre `numpy` è utilizzata per gestire gli array di dati.

6.2 Generazione dei Dati

I dati di esempio sono generati utilizzando la funzione `randn` di `numpy`, che crea un array di 1000 numeri casuali con distribuzione normale:

```
data = np.random.randn(1000)
```

6.3 Creazione della Figura e dell'Asse

Viene creata una figura e un asse per il grafico:

```
fig, ax = plt.subplots()
```

6.4 Creazione dell'Istogramma

L'istogramma viene creato utilizzando la funzione `hist`:

```
n, bins, patches = ax.hist(data, bins=30, color='skyblue',
    edgecolor='black', alpha=0.7)
```

`bins=30` specifica il numero di intervalli, `color='skyblue'` imposta il colore delle barre, `edgecolor='black'` imposta il colore dei bordi delle barre, e `alpha=0.7` imposta la trasparenza delle barre.

6.5 Aggiunta di Etichette e Titolo

Vengono aggiunte etichette agli assi e un titolo al grafico:

```
ax.set_xlabel('Valori')
ax.set_ylabel('Frequenza')
ax.set_title('Istogramma␣dei␣Valori')
```

6.6 Aggiunta di una Legenda e di una Griglia

Infine, vengono aggiunte una legenda e una griglia al grafico:

```
ax.legend(['Dati'])
ax.grid(True)
```

6.7 Visualizzazione del Grafico

Il grafico viene visualizzato con il comando:

```
plt.show()
```

7 Esempio di istogramma sperimentale

7.1 Introduzione

In questo esempio, verranno presentati i dati di un esperimento di misura fisica, la fisica sottostante e il processo per creare un istogramma che rappresenti questi dati utilizzando il linguaggio di programmazione Python con la libreria Matplotlib.

7.2 Descrizione dell'esperimento

L'esperimento consiste nel misurare la lunghezza di un campione di barre metalliche. Le misure sono state effettuate utilizzando un calibro digitale con una precisione di 0.1 mm. Di seguito sono riportati i dati sperimentali ottenuti (in cm):

45.1	47.2	49.3	50.5	52.6	54.7
48.3	46.9	51.2	53.8	50.0	49.9
48.7	51.5	52.1	47.6	46.3	50.9
51.8	48.0	49.5	50.3	47.0	46.5
52.4	48.8	49.2	51.3	47.8	50.7

Nel contesto della statistica, useremo solo programmi software e ci disinteresseremo dei problemi relativi alle cifre significative durante i calcoli, mentre queste saranno importanti nel momento in cui scriveremo il risultato delle misure.

7.3 Fisica dell'esperimento

La misura della lunghezza delle barre metalliche è un esperimento comune in fisica per studiare le proprietà dei materiali. La lunghezza può variare a causa di fattori come:

- Differenze nel processo di produzione.
- Espansione termica a diverse temperature.
- Errori sistematici e casuali durante la misurazione.

L'istogramma delle misure permette di visualizzare la distribuzione delle lunghezze e di identificare eventuali deviazioni significative dalla media.

7.4 Creazione dell'istogramma con Python

Per creare l'istogramma, utilizziamo il seguente codice Python:

```
import matplotlib.pyplot as plt

# Dati sperimentali
dati_sperimentali = [45.1, 47.2, 49.3, 50.5, 52.6, 54.7,
    48.3, 46.9, 51.2, 53.8,
                     50.0, 49.9, 48.7, 51.5, 52.1, 47.6,
                        46.3, 50.9, 51.8, 48.0,
                     49.5, 50.3, 47.0, 46.5, 52.4, 48.8,
                        49.2, 51.3, 47.8, 50.7]

# Crea l'istogramma
plt.figure(figsize=(10, 6))
plt.hist(dati_sperimentali, bins=8, edgecolor='black', alpha
    =0.7)

# Aggiungi titolo e etichette
plt.title('Istogramma dei dati sperimentali')
plt.xlabel('Misura (cm)')
plt.ylabel('Frequenza')

# Mostra l'istogramma
plt.savefig('istogramma2.png')  # Salva l'immagine come
    istogramma2.png
plt.show()
```

Listing 12: Codice Python per creare l'istogramma

7.4.1 Spiegazione del codice

Il codice Python utilizzato per creare l'istogramma è spiegato di seguito:

Importazione delle librerie

```
import matplotlib.pyplot as plt
```

Listing 13: Importazione delle librerie

Questo codice importa la libreria Matplotlib, necessaria per la creazione di grafici.

Definizione dei dati

```
dati_sperimentali = [45.1, 47.2, 49.3, 50.5, 52.6, 54.7,
    48.3, 46.9, 51.2, 53.8,
                     50.0, 49.9, 48.7, 51.5, 52.1, 47.6,
                        46.3, 50.9, 51.8, 48.0,
                     49.5, 50.3, 47.0, 46.5, 52.4, 48.8,
                        49.2, 51.3, 47.8, 50.7]
```

Listing 14: Definizione dei dati

Definisce un array contenente i dati delle misurazioni fisiche.

Creazione dell'istogramma

```
plt.figure(figsize=(10, 6))
plt.hist(dati_sperimentali, bins=8, edgecolor='black', alpha
    =0.7)
```

Listing 15: Creazione dell'istogramma

Imposta la dimensione della figura e crea un istogramma con 8 bin, bordi neri per i bin e trasparenza del 70%.

Aggiunta di titolo e etichette

```
plt.title('Istogramma␣dei␣dati␣sperimentali')
plt.xlabel('Misura␣(cm)')
plt.ylabel('Frequenza')
```

Listing 16: Aggiunta di titolo e etichette

Aggiunge il titolo e le etichette agli assi del grafico.

Visualizzazione e salvataggio dell'istogramma

```
plt.savefig('istogramma2.png')  # Salva l'immagine come
    istogramma2.png
plt.show()
```

Listing 17: Visualizzazione e salvataggio dell'istogramma

Visualizza l'istogramma generato e lo salva come 'istogramma2.png'.

7.5 Risultati

L'istogramma risultante mostra la distribuzione delle lunghezze delle barre metalliche. L'analisi visiva dell'istogramma permette di identificare la variabilità delle misurazioni e la presenza di eventuali outlier.

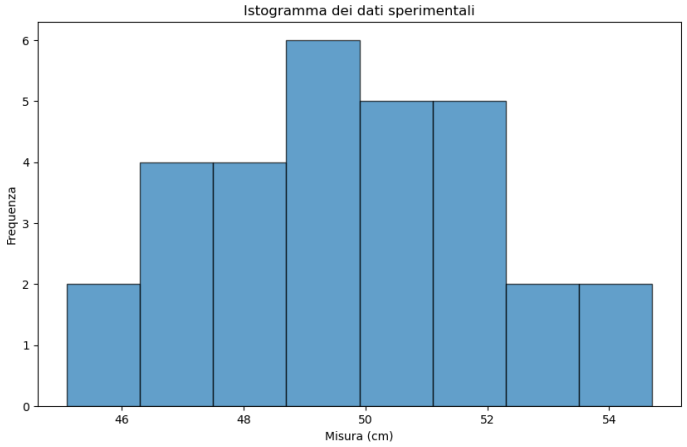

Figura 2: Istogramma dei dati sperimentali

8 Distribuzione di Probabilità

In questa e nelle prossime sezioni, parleremo di probabilità. Quando misuriamo una grandezza casuale, vediamo che alcuni valori si ripetono più frequentemente di altri, ossia abbiamo una *distribuzione* di valori. La statistica ci permette di estrarre da questa distribuzione, informazioni sulla grandezza. Si suppone che ogni grandezza abbia un valore "vero" e che solo effetti casuali discostino il risultato da quest'ultimo. La distribuzione dei valori viene misurata non tanto dalla semidispersione massima (una stima troppo grande) ma dalla deviazione standard (un concetto statistico). Ripetendo le misure molte volte, costruendo un **istogramma delle frequenze**, otteniamo che questo istogramma diventa sempre più liscio e si avvicina ad una curva a forma di campana. In fisica, questa curva si presenta quando ripetiamo molte volte una misura casuale. In quel caso, la grandezza non ha sempre lo stesso valore ma ha, appunto, una *distribuzione*. Se facciamo oscillare un pendolo, e misuriamo la durata t di 20 oscillazioni con un cronometro manuale, non otterremo quasi mai due volte lo stesso valore (a causa di errori umani) ma una serie di valori. La statistica ci consente di prevedere quali saranno i valori più probabili e come questi si distribuiscono, (quanti ad esempio sono molto più

grandi o più piccoli della media). Le prossime sezioni ci insegneranno come trarre informazioni utili dalla distribuzione di queste misure. Impareremo che il risultato di una misura è dato dalla media e dall'errore della media, come segue:

$$x = (\overline{x} \pm \sigma_{\overline{x}}) \text{ u.m.}$$

essendo $\overline{x}$ la media dei valori e $\sigma_{\overline{x}}$ la cosiddetta deviazione standard della media.

8.1 La Curva di Gauss

La distribuzione gaussiana è caratterizzata dalla seguente funzione densità di probabilità (pdf):

$$f(x|\mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{1}$$

dove:

- μ è la media della distribuzione, che indica il valore centrale attorno al quale i dati sono distribuiti.
- σ è la deviazione standard, che misura la dispersione dei dati rispetto alla media.

Non spaventatevi, non importa il significato di tutti i i simboli, non useremo direttamente la formula. A voi basta sapere che questa è una relazione tra quello che c'è a sinistra dell'uguale e quello che c'è a destra. La curva di Gauss ha una forma a campana, simmetrica rispetto alla media μ. La maggior parte dei dati (circa il 68%) si trova entro un intervallo di una deviazione standard dalla media ($\mu \pm \sigma$), mentre il 95% dei dati si trova entro due deviazioni standard ($\mu \pm 2\sigma$). Questo comportamento rende la distribuzione gaussiana particolarmente utile per descrivere fenomeni naturali dove le variazioni sono dovute a molti fattori piccoli e indipendenti. In figura 3 vediamo il tipico aspetto di una gaussiana.

9 Stima di Media e Deviazione Standard

Data una serie di n misurazioni sperimentali $\{x_1, x_2, \ldots, x_n\}$, possiamo stimare i parametri della distribuzione gaussiana, cioè la media μ e la deviazione standard σ, utilizzando le seguenti formule:

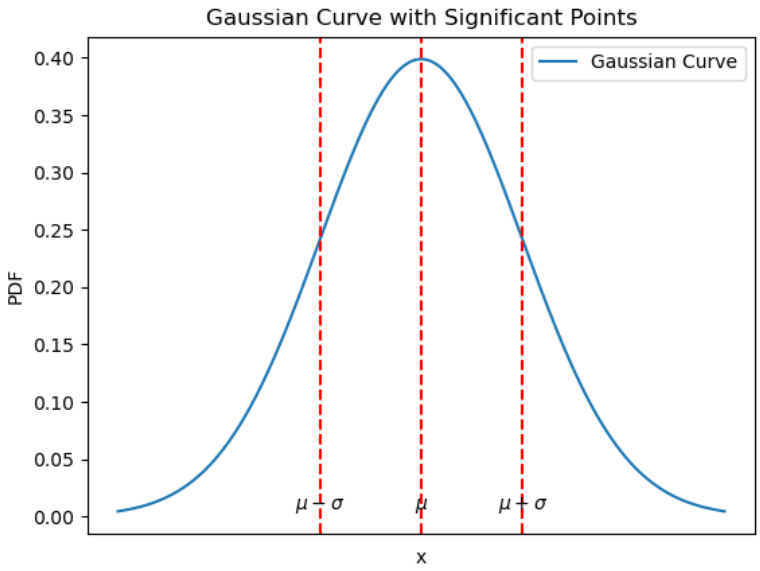

Figura 3: Curva Gaussiana con Evidenziati $\overline{x}$, $\overline{x} - \sigma$, e $\overline{x} + \sigma$

9.1 Calcolo della Media

La media campionaria $\hat{\mu}$ è data dalla somma di tutte le osservazioni divisa per il numero totale di osservazioni:

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{2}$$

9.2 Calcolo della Deviazione Standard

La deviazione standard campionaria $\hat{\sigma}$ è data dalla radice quadrata della somma dei quadrati delle differenze tra ciascuna osservazione e la media campionaria, divisa per il numero di osservazioni meno uno:

$$\hat{\sigma} = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \hat{\mu})^2} \tag{3}$$

9.3 Calcolo della Deviazione Standard della Media

La deviazione standard della media, nota anche come errore standard della media, è calcolata come:

$$\sigma_{\overline{x}} = \frac{\hat{\sigma}}{\sqrt{n}} \tag{4}$$

Dove $\hat{\sigma}$ è la deviazione standard campionaria e n è il numero di osservazioni. La deviazione standard della media rappresenta quanto la media campionaria è attesa essere distante dalla media vera della popolazione. Per una stima più precisa, questa deviazione standard della media viene approssimata a una cifra significativa.

10 Esempio: istogramma e distribuzione

Per illustrare questi concetti, consideriamo un esempio pratico in Python. Abbiamo misurato le altezze di 100 persone (in cm) e calcoleremo la media, la deviazione standard e la deviazione standard della media di queste misure. Successivamente, creeremo un istogramma delle altezze e lo sovrapporremo con la curva gaussiana corrispondente.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

# Altezze di 100 persone (in cm)
heights = [170, 165, 180, 175, 160, 155, 178, 172, 168, 169,
           174, 167, 166, 171, 173, 177, 182, 181, 176, 179,
           164, 163, 162, 161, 159, 158, 157, 156, 154, 153,
           152, 151, 150, 149, 148, 147, 146, 145, 144, 143,
           150, 155, 160, 165, 170, 175, 180, 185, 190, 195,
           172, 177, 182, 187, 192, 197, 162, 167, 172, 177,
           180, 175, 170, 165, 160, 155, 150, 145, 140, 135,
           142, 147, 152, 157, 162, 167, 172, 177, 182, 187,
           165, 170, 175, 180, 185, 190, 195, 200, 205, 210]

# Stima di media, deviazione standard e deviazione standard
   della media
mu = np.mean(heights)
sigma = np.std(heights, ddof=1)
sigma_x_mean = sigma / np.sqrt(len(heights))

# Arrotonda la deviazione standard della media e la media
   alle unità
sigma_x_mean_rounded = round(sigma_x_mean)
mu_rounded = round(mu)

print(f"Media delle altezze: {mu_rounded} cm")
print(f"Deviazione standard delle altezze: {sigma:.2f} cm")
```

```
print(f"Deviazione standard della media: {
    sigma_x_mean_rounded} cm")

# Creazione dell'istogramma
count, bins, ignored = plt.hist(heights, bins=6, density=True
    , alpha=0.6, color='g', edgecolor='black')

# Sovrapposizione della curva gaussiana
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = norm.pdf(x, mu, sigma)
plt.plot(x, p, 'k', linewidth=2)
title = "Istogramma delle Altezze e Curva Gaussiana"
plt.title(title)

# Visualizzazione di media e deviazione standard
plt.axvline(mu, color='r', linestyle='dashed', linewidth=1)
plt.text(mu + mu/10, max(p), f'Media: {mu_rounded} cm', color
    ='r')
plt.axvline(mu + sigma, color='b', linestyle='dashed',
    linewidth=1)
plt.axvline(mu - sigma, color='b', linestyle='dashed',
    linewidth=1)
plt.text(mu + sigma + mu/10, max(p)/2, f'Sigma: {sigma:.2f}
    cm', color='b')
plt.text(mu - sigma - mu/10, max(p)/2, f'Sigma: {sigma:.2f}
    cm', color='b')

# Impostazione dei valori sui bins come etichette sull'asse x
bin_labels = [f"{int(bins[i])}" for i in range(len(bins))]
plt.xticks(bins, labels=bin_labels, rotation=45)

# Migliora la disposizione dei sottotitoli e delle etichette
plt.tight_layout()

plt.savefig('istogramma.png')
#se vuoi scaricare il file da google colab decommenta le
    righe:
#from google.colab import files
#files.download('istogramma.png')
plt.show()

# Risultato finale, arrotondato a due cifre significative
print(f"Altezza = ({mu_rounded} ś {sigma_x_mean_rounded}) cm
    ")
```

Listing 18: Script Python per calcolare e visualizzare le altezze

Ecco l'output:

```
Media delle altezze: 167.83333333333334 cm
Deviazione standard delle altezze: 15.80 cm
Deviazione standard della media: 1.67 cm
Altezza = (168 ś 2) cm
```

11 Spiegazione del Codice per mostrare un istogramma e una gaussiana

11.1 Importazione dei Moduli

```
import matplotlib.pyplot as plt
from scipy.stats import norm
import numpy as np
```

Si importano i moduli necessari: `matplotlib.pyplot` per la creazione dei grafici, `scipy.stats.norm` per la distribuzione normale e `numpy` per le operazioni numeriche.

11.2 Definizione dei Dati

```
heights = [170, 165, 180, ... , 210]
```

Viene creata una lista di altezze di 100 persone in centimetri.

11.3 Calcolo delle Statistiche

```
mu = np.mean(heights)
sigma = np.std(heights, ddof=1)
sigma_x_mean = sigma / np.sqrt(len(heights))
```

Si calcolano la media (`mu`), la deviazione standard (`sigma`) e la deviazione standard della media (`sigma_x_mean`) delle altezze.

11.4 Arrotondamenti

```
sigma_x_mean_rounded = round(sigma_x_mean)
mu_rounded = round(mu)
```

La media e la deviazione standard della media vengono arrotondate alle unità.

11.5 Creazione dell'Istogramma

```
count, bins, ignored = plt.hist(heights, bins=6, density=True
    , alpha=0.6, color='g', edgecolor='black')
```

Viene creato un istogramma delle altezze con 6 bins, normalizzato e colorato di verde con bordo nero.

11.6 Sovrapposizione della Curva Gaussiana

```
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = norm.pdf(x, mu, sigma)
plt.plot(x, p, 'k', linewidth=2)
```

Si calcolano i limiti dell'asse x, si genera un array di valori e si calcola la densità di probabilità della distribuzione normale. La curva gaussiana viene sovrapposta all'istogramma in nero.

11.7 Visualizzazione della Media e della Deviazione Standard

```
plt.axvline(mu, color='r', linestyle='dashed', linewidth=1)
plt.text(mu + mu/10, max(p), f'Media: {mu_rounded} cm', color
    ='r')
plt.axvline(mu + sigma, color='b', linestyle='dashed',
    linewidth=1)
plt.axvline(mu - sigma, color='b', linestyle='dashed',
    linewidth=1)
plt.text(mu + sigma + mu/10, max(p)/2, f'Sigma: {sigma:.2f}
    cm', color='b')
plt.text(mu - sigma - mu/10, max(p)/2, f'Sigma: {sigma:.2f}
    cm', color='b')
```

Vengono disegnate linee verticali tratteggiate per indicare la media e la deviazione standard, con annotazioni per i rispettivi valori.

11.8 Impostazione delle Etichette dei Bins

```
bin_labels = [f"{int(bins[i])}" for i in range(len(bins))]
plt.xticks(bins, labels=bin_labels, rotation=45)
```

I valori dei bins vengono impostati come etichette sull'asse x, ruotate di 45 gradi.

11.9 Miglioramento della Disposizione del Grafico

```
plt.tight_layout()
```

Si ottimizza la disposizione dei sottotitoli e delle etichette per evitare sovrapposizioni.

11.10 Salvataggio e Visualizzazione del Grafico

```
plt.savefig('istogramma.png')
plt.show()
```

Il grafico viene salvato come immagine `istogramma.png` e visualizzato a schermo.

11.11 Risultato Finale

```
print(f"Altezza = ({mu_rounded} ś {sigma_x_mean_rounded}) cm
    ")
```

Viene stampato il risultato finale, arrotondato a due cifre significative. Nel

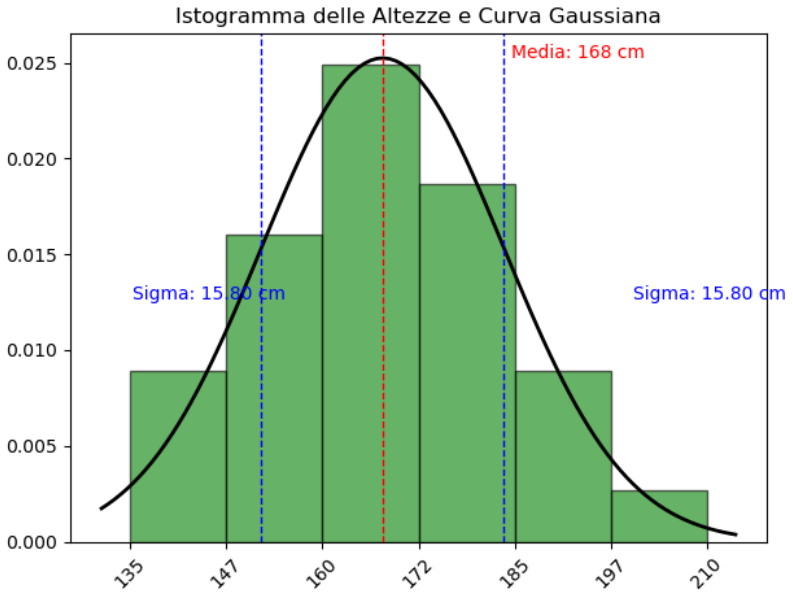

Figura 4: Istogramma delle Altezze con Sovrapposta la Curva Gaussiana

grafico 4 vediamo sovrapposto l'istogramma costruito e la curva che lo approssima. L'istogramma ha in ascisse (asse X) gli intervalli di altezza e in ordinate

(asse Y) un valore tale che l'area del bin sia uguale alla frazione di persone che hanno un'altezza compresa tra i suoi estremi. Ad esempio, se guardiamo l'intervallo tra 160 e 172, l'altezza è 0,025 perché $(172 - 160) \times 0,025 = 0,3$ ossia, il 30% delle persone aveva un'altezza compresa tra 160 e 172 centimetri. Ancora un commento su media e deviazione standard. Notare che la curva è centrata attorno alla media (il valore 167.3 cm) evidenziata dalla linea rossa tratteggiata. Notare anche le due linee blu. La distanza tra la linea verde e quella blu è proprio uguale alla deviazione standard (pari a 15.80 cm e indicata sul grafico come *Sigma*), infatti 167.83 cm + 15.80 cm = 183.63 cm che è proprio dove si trova la linea verde. Notiamo infine che il 68% delle misure capitano tra i valori 167.83 cm − 15.80 cm e 167.83 cm + 15.80 cm, ossia tra 152.03 cm e 183.63 cm. Questo è sempre vero. Quando abbiamo una grandezza che si distribuisce come la curva a campana (la gaussiana scoperta dallo scienziato Gauss) l'area sotto la curva, quella compresa tra le due linee blu, è sempre 0,68 ossia il 68% delle misure che la approssimano, dovrebbero capitare tra $\overline{x} - \sigma$ e $\overline{x} + \sigma$.

12 Cos'è il Best Fit Lineare

In fisica, spesso siamo interessati a comprendere come due grandezze siano correlate tra loro. Un metodo comune per studiare queste relazioni è l'uso del best fit lineare, che ci permette di trovare la retta che meglio approssima i dati sperimentali. In questo documento, esploreremo cos'è il best fit lineare e come può essere utilizzato per analizzare le relazioni tra grandezze fisiche. Il best fit lineare, o regressione lineare, è un metodo statistico utilizzato per determinare la retta che meglio approssima un insieme di dati. Questa retta minimizza la somma dei quadrati delle differenze verticali (chiamate residui) tra i punti dati osservati e i punti previsti dalla retta stessa.

12.1 L'Equazione della Retta

L'equazione di una retta in due dimensioni è:

$$y = b \cdot x + a$$

dove:

- y è la variabile dipendente,
- x è la variabile indipendente,
- b è la pendenza della retta,

- a è l'intercetta della retta con l'asse y.

Questa relazione si incontra spesso in fisica. Un esempio è il moto uniforme. In questo modo, la relazione che lega lo spazio percorso s, la posizione inizile s_0, il tempo impiegato t la velocità v è:

$$s = s_0 + v \cdot t$$

Potremmo disporre di una tabella di dati del tempo e della posizione e volere determinare la velocità e la posizione iniziale col metodo del best fit.

12.2 Calcolo del Best Fit Lineare

Per calcolare il best fit lineare, cioè trovare i parametri a e b e i loro errori, ci appoggeremo a delle librerie Python senza porci il problema di come vengono calcolati. Il problema è complesso e richiede di far variare i due parametri in modo cha la distanza tra la retta e i punti sperimentali, sia la più piccola possibile, ossia la retta passi il più vicino possibile alla maggior parte dei punti sperimentali.

13 Studio di moto rettilineo uniforme

Il moto rettilineo uniforme (MRU) è un movimento con velocità costante. La relazione tra spazio e tempo in un moto rettilineo uniforme può essere descritta dall'equazione:

$$s = s_0 + v \cdot t$$

dove:

- s è la posizione o spazio (che corrisponde a y nella regressione),
- s_0 è la posizione iniziale (che corrisponde all'intercetta a),
- v è la velocità costante (che corrisponde alla pendenza b),
- t è il tempo.

Usando la tecnica del best-fit (o regressione lineare) è possibile determinare posizione iniziale e velocità partendo da un insieme di coppie di dati (tempo, spazio). Nelle prossime sezioni vedremo nel ettaglio il procedimento.

14 Best Fit con `numpy` e `scipy.optimize.curve_fit`

14.1 `numpy` e $a + bx$

La libreria `numpy` è uno strumento potente per il calcolo scientifico in Python. Essa fornisce il supporto per array multidimensionali e una vasta gamma di funzioni matematiche. Nel contesto del best fit, `numpy` è utilizzata per manipolare i dati e applicare operazioni matematiche necessarie per il fitting.

Ad esempio, supponiamo di avere una serie di dati sperimentali che rappresentano due variabili correlate. Utilizzando `numpy`, possiamo rappresentare questi dati come array e prepararli per l'analisi. Le operazioni matematiche e la manipolazione dei dati diventano quindi più efficienti e facili da gestire.

14.2 `scipy` e `curve_fit`

La libreria `scipy` estende le funzionalità di `numpy` fornendo strumenti avanzati per l'ottimizzazione, l'integrazione, l'interpolazione, l'analisi dei segnali e molto altro. In particolare, il modulo `scipy.optimize` contiene la funzione `curve_fit`, uno strumento molto potente per il fitting di curve.

`curve_fit` permette di trovare i parametri di una funzione che meglio approssima i dati sperimentali. Per utilizzare `curve_fit`, si segue un processo in due fasi principali:

1. **Definire la funzione modello**: Questa è la funzione teorica che si pensa possa descrivere i dati. Può trattarsi di una retta, una parabola, una curva esponenziale o qualsiasi altra forma funzionale.

2. **Utilizzare `curve_fit` per trovare i parametri ottimali**: Passando la funzione modello e i dati sperimentali a `curve_fit`, essa restituirà i parametri che meglio approssimano i dati. Questi parametri possono poi essere utilizzati per fare previsioni o per comprendere meglio la relazione tra le variabili.

L'utilizzo congiunto di `numpy` e `scipy.optimize.curve_fit` rende possibile affrontare in modo efficace e robusto il problema del best fit, permettendo un'analisi quantitativa precisa delle relazioni tra variabili sperimentali. Questi strumenti sono essenziali per chiunque lavori con dati numerici e desideri ottenere il massimo delle informazioni dai propri esperimenti.

15 Esempio di Dati Sperimentali

Supponiamo di avere i seguenti dati sperimentali che descrivono la posizione di un oggetto in funzione del tempo. La posizione s è linearmente legata al tempo t con una velocità costante, ma con un piccolo errore random.

Tempo (s)	Spazio (m)
1.0	2.1
2.0	4.2
3.0	6.1
4.0	8.0
5.0	10.1

Tabella 2: Tabella dei dati sperimentali per il moto rettilineo uniforme.

16 Codice Python per la Regressione Lineare

Il seguente codice Python esegue una regressione lineare sui dati sperimentali per determinare la velocità v e la sua incertezza.

```
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

# Dati sperimentali
tempo = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
spazio = np.array([2.1, 4.2, 6.1, 8.0, 10.1])
error_spazio = np.array([0.1, 0.1, 0.1, 0.1, 0.1])


# Definizione della funzione di modello lineare
def linear_model(t, s0, v):
    return s0 + v * t

# Regressione lineare
params, covariance = curve_fit(linear_model, tempo, spazio)
s0 = params[0]
v = params[1]
std_err = np.sqrt(np.diag(covariance))

# Calcolo dell'errore sulla velocità
v_error = std_err[1]

print(f"Velocità (v): {v:.2f} $\pm$ {v_error:.2f} m/s")

# Creazione del grafico
plt.figure(figsize=(8, 6))
plt.errorbar(tempo, spazio, yerr=error_spazio, fmt='o', label
   ='Dati sperimentali', capsize=5)
plt.plot(tempo, linear_model(tempo, *params), 'r-', label='
   Retta di regressione')
plt.title('Moto Rettilineo Uniforme')
plt.xlabel('Tempo (s)')
plt.ylabel('Spazio (m)')
plt.legend()
plt.grid(True)
plt.savefig(   )
#se vuoi scaricare il file da google colab decommenta le
   righe:
#from google.colab import files
#files.download('regressione1.png')
plt.show()
```

Listing 19: Script Python per eseguire la regressione lineare

Il risultato in output è

```
Velocità (v): 1.98 +- 0.02 m/s
```

mentre il grafico è mostrato nella figura 5

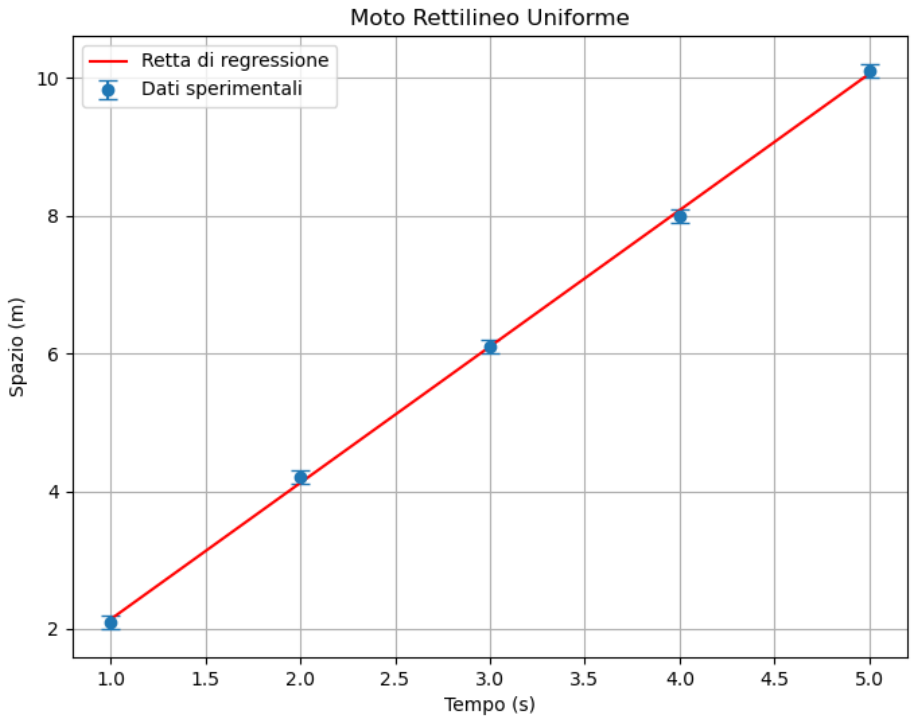

Figura 5: Best fit di un moto uniforme.

17 Spiegazione dei Risultati

Il codice Python utilizza la funzione `curve_fit` del modulo `scipy.optimize` per eseguire una regressione lineare sui dati. La funzione di modello `linear_model` rappresenta l'equazione del moto rettilineo uniforme: dove s_0 è l'intercetta e v è la velocità. La regressione lineare determina i valori ottimali per s_0 e v minimizzando la somma dei quadrati degli errori tra i dati osservati e i dati previsti dal modello. Il grafico risultante mostra i dati sperimentali con le barre d'errore e la retta di regressione, che rappresenta la relazione lineare tra spazio e tempo. Il grafico di output è nella figura 5. Questo codice si

può adattare facilmente ad altre situazioni sperimentali cambiando i dati e il nome delle grandezze (per una molla potremmo avere al posto di t ed s la lunghezza e la forza ad esempio).

17.1 Esempio complesso

Supponiamo di avere i dati sperimentali nella tabella 3 che rappresenta i tempi di caduta di un oggetto da diverse altezze:

Tempo (s)	Distanza (m)
1.0	4.9
2.0	19.6
3.0	44.1
4.0	78.4
5.0	122.5

Tabella 3: Dati di un moto di caduta.

17.2 Analisi dei Dati

Prima di applicare il best fit lineare, calcoliamo il tempo al quadrato t^2. Infatti la formula del moto di caduta:

$$s = \frac{1}{2} g \cdot t^2$$

ci dice che se pongo $t^2 = t2$, la relazione tra s e $t2$ è lineare perché la formula si scrive:

$$s = \frac{1}{2} g \cdot t2$$

mentre quella tra t ed s è quadratica. A questo punto, se poniamo $b = \frac{g}{2}$, la relazione si può scrivere:

$$s = b \cdot 2t$$

essendo quindi una relazione senza parametro intercetta. Una volta trovato b, determiniamo l'accelerazione di gravità da una formula inversa.

$$g = 2 \cdot b$$

e il suo errore propagato vale:

$$\sigma_g = 2 \cdot \sigma_b$$

essendo σ_b l'errore sul parametro b (nel codice, gli errori sulle y, sul parametro a e sul parametro b si ottengono dal vettorem

Tempo (s)	Tempo al quadrato (s^2)
1.0	1.0
2.0	4.0
3.0	9.0
4.0	16.0
5.0	25.0

Il codice Python per eseguire la regressione lineare e calcolare l'accelerazione di gravità è:

```
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

# Dati sperimentali
tempo = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
distanza = np.array([4.9, 19.6, 44.1, 78.4, 122.5])
error_distanza = np.array([0.1, 0.1, 0.1, 0.1, 0.1])

# Calcolo di t^2
tempo_squared = tempo**2

# Definizione della funzione di modello lineare senza
    intercetta
def linear_model(t_squared, g_over_2):
    return g_over_2 * t_squared

# Regressione lineare forzata attraverso l'origine
params, covariance = curve_fit(linear_model, tempo_squared,
    distanza, sigma=error_distanza, absolute_sigma=True)
g_over_2 = params[0]

# Calcolo dell'accelerazione di gravità
g = 2 * g_over_2

# Calcolo dell'errore associato
std_err = np.sqrt(np.diag(covariance))
g_error = 2 * std_err[0]


print(f"Accelerazione di gravità (g): {g:.2f} ś {g_error:.2f}
    m/s^2")

# Creazione del grafico
plt.figure(figsize=(8, 6))
plt.errorbar(tempo_squared, distanza, yerr=error_distanza,
    fmt='o', label='Dati sperimentali', capsize=5)
plt.plot(tempo_squared, linear_model(tempo_squared, *params),
    'r-', label='Retta di regressione')
```

```
plt.title('Misurazione dell\'Accelerazione di Gravità')
plt.xlabel('Tempo al quadrato (s^2)')
plt.ylabel('Distanza (m)')
plt.legend()
plt.grid(True)
plt.savefig('regressione2.png')
#se vuoi scaricare il file da google colab decommenta le
   righe:
#from google.colab import files
#files.download('regressione2.png')
plt.show()
```

Listing 20: Calcolo della Regressione Lineare e Accelerazione di Gravità

Il grafico è mostrato in figura.6. L'output dello script è:

```
Accelerazione di gravità (g): 9.80 +- 0.01 m/s^2
```

Segue spiegazione del codice.

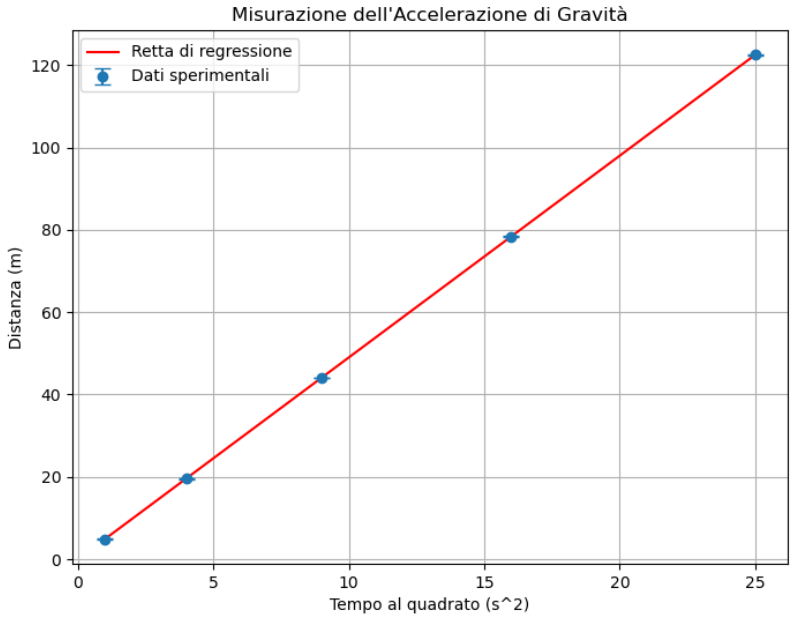

Figura 6: Grafico della regressione lineare per il calcolo dell'accelerazione dalla pendenza.

18 Dettagli sul codice

18.1 Dati Sperimentali

I dati di tempo e distanza sono memorizzati nei seguenti array:

- `tempo`: Array contenente i valori di tempo (in secondi).
- `distanza`: Array contenente i valori di distanza (in metri) misurati.
- `error_distanza`: Array contenente gli errori associati alle misure di distanza (in metri).

18.2 Calcolo di t^2

Per adattare il modello $d = \frac{1}{2}gt^2$, è necessario calcolare il tempo al quadrato. Questo viene fatto con:

```
# Calcolo del tempo al quadrato
tempo_squared = tempo**2
```

Il vettore `tempo_squared` contiene i valori del tempo al quadrato, che sono usati per la regressione lineare.

18.3 Definizione della Funzione di Modello

Definiamo una funzione lineare che rappresenta la relazione tra distanza e tempo al quadrato:

```
# Definizione della funzione lineare
def linear_model(t_squared, g_over_2):
    return g_over_2 * t_squared
```

Questa funzione modella la relazione $d = \frac{1}{2}gt^2$ e assume che l'intercetta sia zero. `g_over_2` rappresenta la metà dell'accelerazione di gravità.

18.4 Regressione Lineare

Utilizziamo la funzione `curve_fit` per eseguire la regressione lineare. I parametri e la covarianza sono ottenuti da:

```
# Regressione lineare forzata attraverso l'origine
params, covariance = curve_fit(linear_model, tempo_squared,
   distanza, sigma=error_distanza, absolute_sigma=True)
g_over_2 = params[0]
```

- `curve_fit`: Funzione che calcola i parametri migliori per adattare il modello ai dati.
- `linear_model`: La funzione di modello lineare definita in precedenza.
- `tempo_squared`: Array contenente i valori di tempo al quadrato.
- `distanza`: Array contenente i valori di distanza misurati.
- `sigma=error_distanza`: Specifica gli errori associati alle misure di distanza.
- `absolute_sigma=True`: Indica che gli errori forniti sono errori assoluti.
- `params`: Array contenente i parametri stimati (in questo caso, `g_over_2`).
- `covariance`: Matrice di covarianza dei parametri stimati, utilizzata per calcolare l'incertezza associata.

18.5 Calcolo dell'Accelerazione di Gravità

L'accelerazione di gravità g è calcolata come:

```
# Calcolo di g
g = 2 * g_over_2
```

Poiché il nostro modello è $d = \frac{1}{2}gt^2$, la pendenza g_over_2 è la metà dell'accelerazione di gravità.

18.6 Calcolo dell'Errore Associato

L'errore associato a g è calcolato utilizzando la covarianza dei parametri:

```
# Calcolo dell'errore associato
std_err = np.sqrt(np.diag(covariance))
g_error = 2 * std_err[0]
```

- `np.sqrt(np.diag(covariance))`: Calcola l'errore standard dei parametri stimati.
- `std_err`: Array contenente gli errori standard.
- `g_error`: L'errore associato all'accelerazione di gravità g, calcolato come il doppio dell'errore della pendenza $g/2$.

18.7 Creazione del Grafico

Il grafico visualizza i dati sperimentali e la retta di regressione. Le barre d'errore sono mostrate per indicare l'incertezza nella misura della distanza:

```
# Creazione del grafico
plt.figure(figsize=(8, 6))
plt.errorbar(tempo_squared, distanza, yerr=error_distanza,
   fmt='o', label='Dati sperimentali', capsize=5)
plt.plot(tempo_squared, linear_model(tempo_squared, *params),
    'r-', label='Retta di regressione')
plt.title('Misurazione dell\'Accelerazione di Gravità')
plt.xlabel('Tempo al quadrato (s^2)')
plt.ylabel('Distanza (m)')
plt.legend()
plt.grid(True)
plt.show()
```

- `plt.figure(figsize=(8, 6))`: Crea una nuova figura di dimensioni 8x6 pollici.
- `plt.errorbar`: Crea un grafico a dispersione con barre d'errore per ogni punto dati.
- `tempo_squared`: Array con i valori di tempo al quadrato.
- `distanza`: Array con i valori di distanza misurati.
- `yerr=error_distanza`: Specifica gli errori associati alle misure di distanza.
- `fmt='o'`: Specifica che i punti dati sono rappresentati come cerchi.
- `label='Dati sperimentali'`: Etichetta per i dati sperimentali.
- `capsize=5`: Lunghezza delle barre orizzontali alle estremità delle barre d'errore.
- `plt.plot`: Disegna la retta di regressione.
- `linear_model(tempo_squared, *params)`: Valori previsti dal modello lineare.
- `'r-'`: Specifica che la retta di regressione è rossa e solida.
- `label='Retta di regressione'`: Etichetta per la retta di regressione.

- `plt.title`: Aggiunge il titolo al grafico.
- `plt.xlabel`: Aggiunge l'etichetta all'asse delle ascisse.
- `plt.ylabel`: Aggiunge l'etichetta all'asse delle ordinate.
- `plt.legend()`: Mostra la legenda nel grafico.
- `plt.grid(True)`: Aggiunge una griglia al grafico.
- `plt.show()`: Visualizza il grafico.