

Relazione di Laboratorio - Guidovia

Francesco Forcher

Matricola: 1073458

`mailto:francesco.forcher@studenti.unipd.it`

Francesca Damiani

Matricola: 1071072

`mailto:francesca.damiani@studenti.unipd.it`

Andrea Piccinin

Matricola: 1070620

`mailto:andrea.piccinin1@studenti.unipd.it`

28 febbraio 2014

Indice

1	Obiettivi	3
2	Descrizione dell'apparato strumentale	3
3	Metodologia di misura	3
4	Presentazione dati sperimentali	4
4.1	Inclinazione 15', senza peso	4
4.2	Inclinazione 30', senza peso	5
4.3	Inclinazione 45', senza peso	6
4.4	Inclinazione 45', con peso	7
4.5	Inclinazione 0', senza peso, senza spessore	8
4.6	Inclinazione 0', senza peso, con spessore	9
4.7	Inclinazione 0', con peso, senza spessore	10
4.8	Inclinazione 0', con peso, con spessore	11
5	Discussione dati sperimentali	12
5.1	Tabella: coefficienti angolari	12
5.2	Tabella: stime accelerazioni di gravità	13
6	Conclusioni	14
7	Grafici	15
7.1	Inclinazione 15', senza peso	15
7.2	Inclinazione 30', senza peso	16
7.3	Inclinazione 45', senza peso	17
7.4	Inclinazione 45', con peso	18
7.5	Inclinazione 0', senza peso, senza spessore	19
7.6	Inclinazione 0', senza peso, con spessore	20
7.7	Inclinazione 0', con peso, senza spessore	21
7.8	Inclinazione 0', con peso, con spessore	22
8	Codice	23

1 Obiettivi

Stimare il valore dell'accelerazione di gravità g attraverso la misurazione dell'accelerazione della slitta su un piano inclinato, tenendo conto, nella seconda parte dell'esperienza, anche del contributo dell'attrito dell'aria.

2 Descrizione dell'apparato strumentale

Slitta in plexiglass che, inizialmente bloccata da un elettromagnete, scorre lungo una guidovia di acciaio, il cui attrito con la slitta è rimosso da un cuscino d'aria. Traguardi mobili a sensori infrarossi, collegati ad un cronometro di sensibilità 10^{-3} s. Nella seconda parte dell'esperienza, essendo la guida posta orizzontalmente, la slitta è stata fatta muovere attraverso un impulso elettrico dato dall'elettromagnete, che ne determina la velocità iniziale.

3 Metodologia di misura

Nella prima parte dell'esperienza, l'elettromagnete che trattiene la slitta viene disattivato tramite un pulsante. La slitta inizia così a scorrere lungo la guidovia su cui sono posizionati i traguardi, che inviano al cronometro il tempo di percorrenza dell'intervallo stabilito. Le misure vengono prese partendo da 40 cm dall'elettromagnete, aumentando l'ampiezza dell'intervallo da 10 cm a 70 cm. Le misure vengono poi ripetute applicando un disco di ottone sulla slitta in modo da modificarne il peso. Nella seconda parte dell'esperienza, essendo la guidovia orizzontale, si fa muovere la slitta attraverso un impulso elettrico, che ne determina la velocità iniziale. In questo caso le misure vengono prese su intervalli di 20 cm, sempre partendo da 40 cm di distanza dall'elettromagnete, in modo da stimare una riduzione della velocità dovuta alla forza di attrito. La velocità iniziale viene poi modificata interponendo tra la slitta e l'elettromagnete uno spessore in alluminio.

4 Presentazione dati sperimentali

Riportiamo in seguito le misure tabulate, con relative statistiche.

4.1 Inclinazione 15', senza peso

La quarta misura nell'intervallo 40-60 cm ha un valore non compatibile con gli altri. Potrebbe esserci stato un errore durante la misurazione. Ma adottando il modello di selezione dei dati oggettivo dei 3 sigma, poichè il valore rientra nei limiti, è stato lasciato.

Intervalli (cm)	40 – 50	40 – 60	40 – 70	40 – 80	40 – 90	40 – 100	40 – 110
Misure (s)	0.648	1.200	1.714	2.174	2.594	2.988	3.348
	0.644	1.206	1.701	2.157	2.582	2.992	3.384
	0.642	1.198	1.700	2.170	2.578	2.978	3.380
	0.639	1.804	1.706	2.157	2.586	2.976	3.354
	0.644	1.200	1.697	2.162	2.588	2.980	3.361
Media (s)	0.643	1.3	1.704	2.164	2.586	2.983	3.365
Varianza del campione (s^2)	0.001	0.05	0.001	0.001	0.001	0.001	0.001
Deviazione standard campione (s)	0.003	0.2	0.006	0.007	0.005	0.006	0.01
Varianza della popolazione (s^2)	0.001	0.07	0.001	0.001	0.001	0.001	0.001
Deviazione standard popolazione (s)	0.003	0.3	0.007	0.008	0.006	0.007	0.02
Errore della media (s)	0.001	0.1	0.003	0.003	0.003	0.003	0.007
Massimo (s)	0.648	1.8	1.714	2.174	2.594	2.992	3.384
Minimo (s)	0.639	1.1	1.697	2.157	2.578	2.976	3.348

4.2 Inclinazione 30', senza peso

Intervalli (cm)	40 – 50	40 – 60	40 – 70	40 – 80	40 – 90	40 – 100	40 – 110
Misure (s)	0.453	0.848	1.196	1.514	1.819	2.088	2.350
	0.452	0.846	1.200	1.518	1.822	2.093	2.350
	0.454	0.850	1.196	1.520	1.812	2.093	2.346
	0.452	0.849	1.196	1.522	1.812	2.092	2.350
	0.452	0.846	1.192	1.514	1.811	2.099	2.354
Media (s)	0.453	0.848	1.196	1.518	1.815	2.093	2.350
Varianza del campione (s^2)	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Deviazione standard campione (s)	0.001	0.002	0.003	0.003	0.004	0.004	0.003
Varianza della popolazione (s^2)	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Deviazione standard popolazione (s)	0.001	0.002	0.003	0.004	0.005	0.004	0.003
Errore della media (s)	0.001	0.001	0.001	0.002	0.002	0.002	0.001
Massimo (s)	0.454	0.850	1.200	1.522	1.822	2.099	2.354
Minimo (s)	0.452	0.846	1.192	1.514	1.811	2.088	2.346

4.3 Inclinazione 45', senza peso

Intervalli (cm)	40 – 50	40 – 60	40 – 70	40 – 80	40 – 90	40 – 100	40 – 110
Misure (s)	0.367	0.688	0.974	1.236	1.477	1.704	1.908
	0.368	0.687	0.972	1.235	1.478	1.702	1.912
	0.367	0.687	0.974	1.235	1.470	1.703	1.906
	0.368	0.690	0.974	1.234	1.480	1.702	1.910
	0.368	0.688	0.973	1.235	1.481	1.700	1.912
Media (s)	0.368	0.688	0.973	1.235	1.477	1.702	1.910
Varianza del campione (s^2)	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Deviazione standard campione (s)	0.001	0.001	0.001	0.001	0.004	0.001	0.002
Varianza della popolazione (s^2)	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Deviazione standard popolazione (s)	0.001	0.001	0.001	0.001	0.004	0.001	0.003
Errore della media (s)	0.000	0.001	0.001	0.001	0.002	0.001	0.001
Massimo (s)	0.368	0.690	0.974	1.236	1.481	1.704	1.912
Minimo (s)	0.367	0.687	0.972	1.234	1.470	1.700	1.906

4.4 Inclinazione 45', con peso

Intervalli (cm)	40 – 50	40 – 60	40 – 70	40 – 80	40 – 90	40 – 100	40 – 110
Misure (s)	0.367	0.688	0.975	1.234	1.435	1.702	1.912
	0.368	0.688	0.976	1.236	1.474	1.700	1.912
	0.368	0.690	0.975	1.234	1.474	1.700	1.912
	0.368	0.690	0.974	1.234	1.474	1.700	1.913
	0.368	0.690	0.974	1.234	1.476	1.700	1.914
Media (s)	0.368	0.689	0.975	1.234	1.467	1.700	1.913
Varianza del campione (s^2)	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Deviazione standard campione (s)	0.001	0.001	0.001	0.001	0.02	0.001	0.001
Varianza della popolazione (s^2)	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Deviazione standard popolazione (s)	0.001	0.001	0.001	0.001	0.02	0.001	0.001
Errore della media (s)	0.001	0.001	0.001	0.001	0.008	0.001	0.001
Massimo (s)	0.368	0.690	0.976	1.236	1.476	1.702	1.914
Minimo (s)	0.367	0.688	0.974	1.234	1.435	1.700	1.912

4.5 Inclinazione 0', senza peso, senza spessore

Intervalli (cm)	40 – 60	50 – 70	60 – 80	70 – 90	80 – 100	90 – 110
Misure (s)	1.168	1.182	1.187	1.194	1.207	1.160
	1.154	1.160	1.188	1.186	1.184	1.174
	1.158	1.172	1.187	1.196	1.192	1.168
	1.162	1.186	1.176	1.190	1.208	1.182
	1.156	1.177	1.185	1.200	1.196	1.172
Media (s)	1.160	1.175	1.185	1.193	1.197	1.171
Varianza del campione (s^2)	0.001	0.001	0.001	0.001	0.001	0.001
Deviazione standard campione (s)	0.005	0.009	0.004	0.005	0.009	0.007
Varianza della popolazione (s^2)	0.001	0.001	0.001	0.001	0.001	0.001
Deviazione standard popolazione (s)	0.006	0.01	0.005	0.005	0.01	0.008
Errore della media (s)	0.002	0.005	0.002	0.002	0.005	0.004
Massimo (s)	1.168	1.186	1.188	1.2	1.208	1.182
Minimo (s)	1.154	1.16	1.176	1.186	1.184	1.16

4.6 Inclinazione 0', senza peso, con spessore

Intervalli (cm)	40 – 60	50 – 70	60 – 80	70 – 90	80 – 100	90 – 110
Misure (s)	1.664	1.660	1.728	1.705	1.724	1.662
	1.642	1.666	1.758	1.697	1.774	1.681
	1.668	1.680	1.736	1.723	1.772	1.702
	1.630	1.668	1.706	1.726	1.714	1.741
	1.637	1.666	1.726	1.760	1.752	1.710
Media (s)	1.648	1.668	1.731	1.722	1.747	1.699
Varianza del campione (s^2)	0.001	0.001	0.001	0.001	0.001	0.001
Deviazione standard campione (s)	0.02	0.007	0.02	0.02	0.02	0.03
Varianza della popolazione (s^2)	0.001	0.001	0.001	0.001	0.001	0.001
Deviazione standard popolazione (s)	0.02	0.007	0.02	0.02	0.03	0.03
Errore della media (s)	0.008	0.003	0.008	0.01	0.01	0.01
Massimo (s)	1.668	1.680	1.758	1.760	1.774	1.741
Minimo (s)	1.630	1.660	1.706	1.697	1.714	1.662

4.7 Inclinazione 0', con peso, senza spessore

Intervalli (cm)	40 – 60	50 – 70	60 – 80	70 – 90	80 – 100	90 – 110
Misure (s)	1.708	1.761	1.838	1.889	1.927	1.939
	1.720	1.764	1.815	1.848	1.896	1.961
	1.720	1.772	1.840	1.870	1.930	1.968
	1.730	1.777	1.822	1.856	1.970	1.939
	1.733	1.777	1.808	1.880	1.896	1.900
Media (s)	1.722	1.770	1.825	1.869	1.924	1.941
Varianza del campione (s^2)	0.001	0.001	0.001	0.001	0.001	0.001
Deviazione standard campione (s)	0.009	0.007	0.01	0.02	0.03	0.02
Varianza della popolazione (s^2)	0.001	0.001	0.001	0.001	0.001	0.001
Deviazione standard popolazione (s)	0.01	0.007	0.01	0.02	0.03	0.03
Errore della media (s)	0.004	0.003	0.006	0.008	0.01	0.01
Massimo (s)	1.733	1.777	1.840	1.889	1.970	1.968
Minimo (s)	1.708	1.761	1.808	1.848	1.896	1.900

4.8 Inclinazione 0', con peso, con spessore

Intervalli (cm)	40 – 60	50 – 70	60 – 80	70 – 90	80 – 100	90 – 110
Misure (s)	2.678	2.814	3.036	3.164	3.41	3.774
	2.66	2.777	2.962	3.248	3.269	3.425
	2.654	2.804	3.028	3.067	3.384	3.845
	2.664	2.844	2.953	3.188	3.244	3.692
	2.652	2.812	2.986	3.146	3.293	3.748
Media (s)	2.662	2.810	2.993	3.163	3.320	3.697
Varianza del campione (s^2)	0.001	0.001	0.001	0.003	0.004	0.02
Deviazione standard campione (s)	0.009	0.02	0.03	0.06	0.07	0.1
Varianza della popolazione (s^2)	0.001	0.001	0.001	0.004	0.005	0.03
Deviazione standard popolazione (s)	0.01	0.02	0.04	0.07	0.07	0.2
Errore della media (s)	0.005	0.01	0.02	0.03	0.03	0.07
Massimo (s)	2.678	2.844	3.036	3.248	3.410	3.845
Minimo (s)	2.652	2.777	2.953	3.067	3.244	3.425

5 Discussione dati sperimentali

Prima di discutere i dati sperimentali si noti che si è deciso di calcolare le velocità del carrello scorrevole lungo la guidovia, facendo la media dei tempi misurati sperimentalmente, e poi utilizzando questo valore medio per calcolare le velocità media, e non calcolando un valore di velocità per ogni tempo registrato, per poi calcolare la media di queste velocità, perchè con quest'ultimo metodo si è dimostrato che l'errore propagato è maggiore rispetto al primo metodo. In seguito vengono allegate due tabelle rappresentanti un elaborazione delle misure prese durante le esperienze:

5.1 Tabella: coefficienti angolari

COEFFICIENTI ANGOLARI DELLE RETTE INTERPOLANTI		
$1^a \text{ Esperienza : } v = a + bt$		
Pendenza: $b \ [m * s^{-2}]$	Errore quadratico medio $[m * s^{-2}]$	Tipo di Campione
0.0389	0.0006	15', no peso
0.081	0.002	30', no peso
0.122	0.002	45', no peso
0.123	0.001	45', si peso
$2^a \text{ Esperienza : } v = a + bx$		
Pendenza: $b \ [s^{-1}]$	Errore quadratico medio $[s^{-1}]$	Tipo di Campione
-0.007	0.005	0', no peso, no spessore
-0.014	0.005	0', no peso, spessore
-0.028	0.002	0', peso, no spessore
-0.040	0.001	0', peso, spessore

5.2 Tabella: stime accelerazioni di gravità

ACCELERAZIONI di GRAVITA'		
Accelerazione $g \ [m * s^{-2}]$	Errore quadratico medio $[m * s^{-2}]$	Tipo di Campione
8.9	0.2	15', no peso
9.3	0.2	30', no peso
9.3	0.2	45', no peso
9.34	0.09	45', si peso
FATTORI di CORREZIONE delle ACCELERAZIONI DI GRAVITA'		
0.010	0.003	Dati senza peso
0.0339	0.0009	Dati con peso

Nella prima tabella vengono esposti tutti i valori dei coefficienti angolari, con il relativo errore, estrapolati (tramite il programma grafico: Gnuplot) dalle rette illustrate nelle precedenti rappresentazioni; Nella

seconda tabella invece vengono mostrate le stime dell'accelerazione di gravità (nella sezione riguardante la prima parte dell'esperienza) e i loro fattori di correzione (nella sezione riguardante la seconda parte dell'esperienza). Le stime dell'accelerazione di gravità, \mathbf{g} , ricavate dalla prima serie di esperimenti, ovvero lavorando con la guidovia a varie inclinazioni (15', 30', 45' con e senza peso) forniscono una stima dell'accelerazione di gravità media, \mathbf{g}_0 , pari a $(9.21 \pm 0.08) \text{ m}\cdot\text{s}^{-2}$. Già a prima vista si può notare che esso differisce di molto rispetto al valore atteso a padova, $\mathbf{g}_p = (9.806 \pm 0.001) \text{ m}\cdot\text{s}^{-2}$. Matematicamente quest'osservazione è confermata dal valore della compatibilità tra le due misure, ovvero 7.45; Queste sono chiaramente incompatibili. Probabilmente ciò è dovuto a causa degli errori sperimentali commessi durante l'esecuzione dell'esperimento dagli operatori. Il valore appena calcolato \mathbf{g}_0 è comunque soggetto ad un errore dovuto alle forze di attrito agenti sull'apparato sperimentale. Questo errore è stato corretto di un fattore $\Delta\mathbf{g}$, calcolato nella seconda serie di esperimenti, in cui la guidovia è stata tenuta con inclinazione nulla. La media dei quattro valori di correzione (no peso, no spessore; no peso, spessore; peso, no spessore; peso, spessore) è pari a: **$(9.21266 \pm 0.0840055) \text{ m}\cdot\text{s}^{-2}$** ; per cui la nostra stima dell'accelerazione di gravità, corretta dalle forze di attrito, è pari a $\mathbf{g} = \mathbf{g}_0 + \Delta\mathbf{g} = (9.42 \pm 0.09) \text{ m}\cdot\text{s}^{-2}$. La compatibilità di questo valore rispetto al valore atteso a padova è pari a: 4.29, ovvero è ancora non compatibile, nonostante, com'è lecito aspettarsi, sia comunque una miglior stima rispetto al valore non corretto dalle forze di attrito.

6 Conclusioni

Per verificare quale sia il metodo migliore per stimare l'accelerazione di gravità si allega questa tabella in cui sono riassunti tutti i valori di \mathbf{g} , corretti dall'errore dovuto all'attrito, correlati alla relativa compatibilità rispetto al valore atteso a padova, \mathbf{g}_p .

STIME g CORRETTE				
Tipo di Dato	g [m/s ²] corretto	g [m/s ²]	Compatibilità	
15'	9.3	0.3	1.69	
30'	9.5	0.2	1.53	
45'	9.4	0.2	2.03	
45' peso	9.57	0.09	2.62	

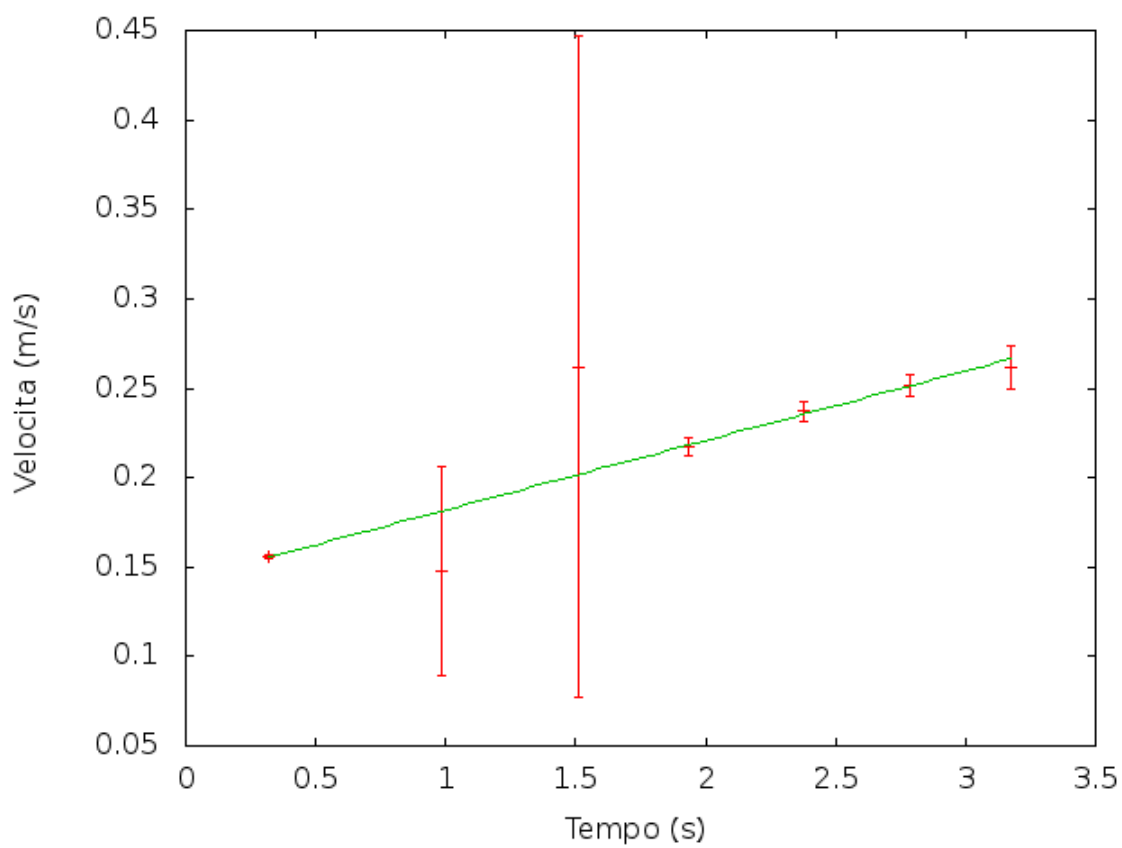
Questa tabella mostra delle particolarità, ovvero il valore con la miglior compatibilità rispetto al valore atteso è quello relativo ad un'inclinazione di 15' della guidovia, e senza il peso. Ciononostante si nota che il suo errore è parecchio elevato, al contrario del valore relativo all'inclinazione 45', con il peso, questo infatti ha una cattiva compatibilità, ma un errore molto basso. Si deduce da questo che il metodo più preciso per calcolare l'accelerazione di gravità è una maggior inclinazione dell'apparato sperimentale con applicato al

carrello scorrevole un peso (Situazione sperimentale della guidovia: 45', Applicando il peso). Si ipotizza che l'elevata compatibilità già evidenziata sopra (il che contraddice la mia ipotesi riguardo il metodo più preciso) sia dovuta al grande errore della misura stessa.

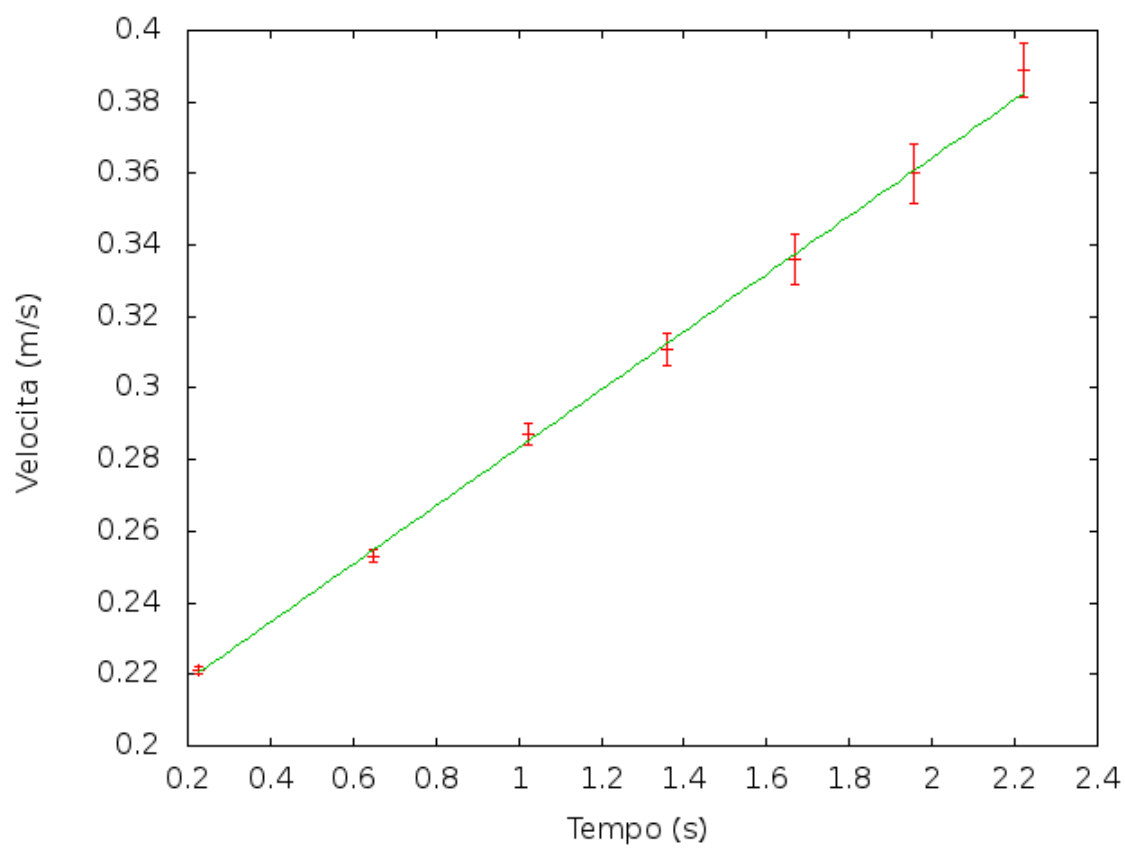
7 Grafici

Riportiamo in seguito i grafici delle velocità medie e le rette interpolanti.

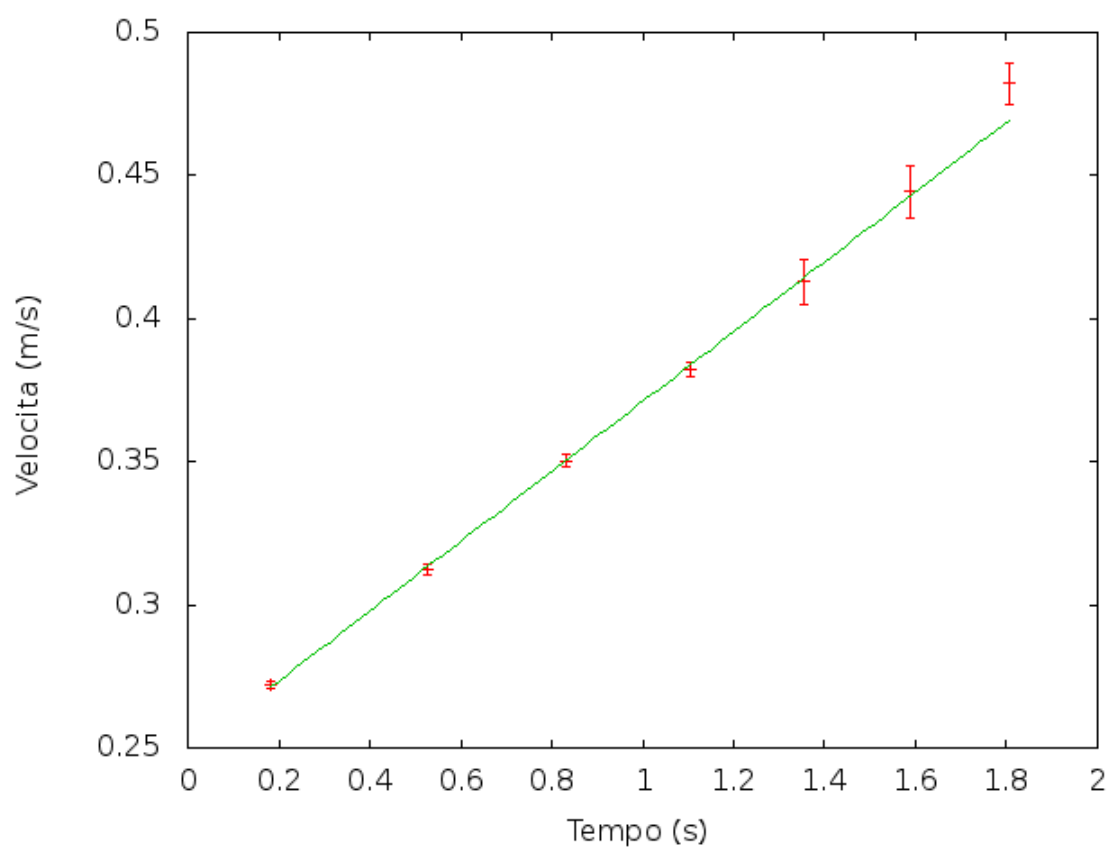
7.1 Inclinazione 15', senza peso



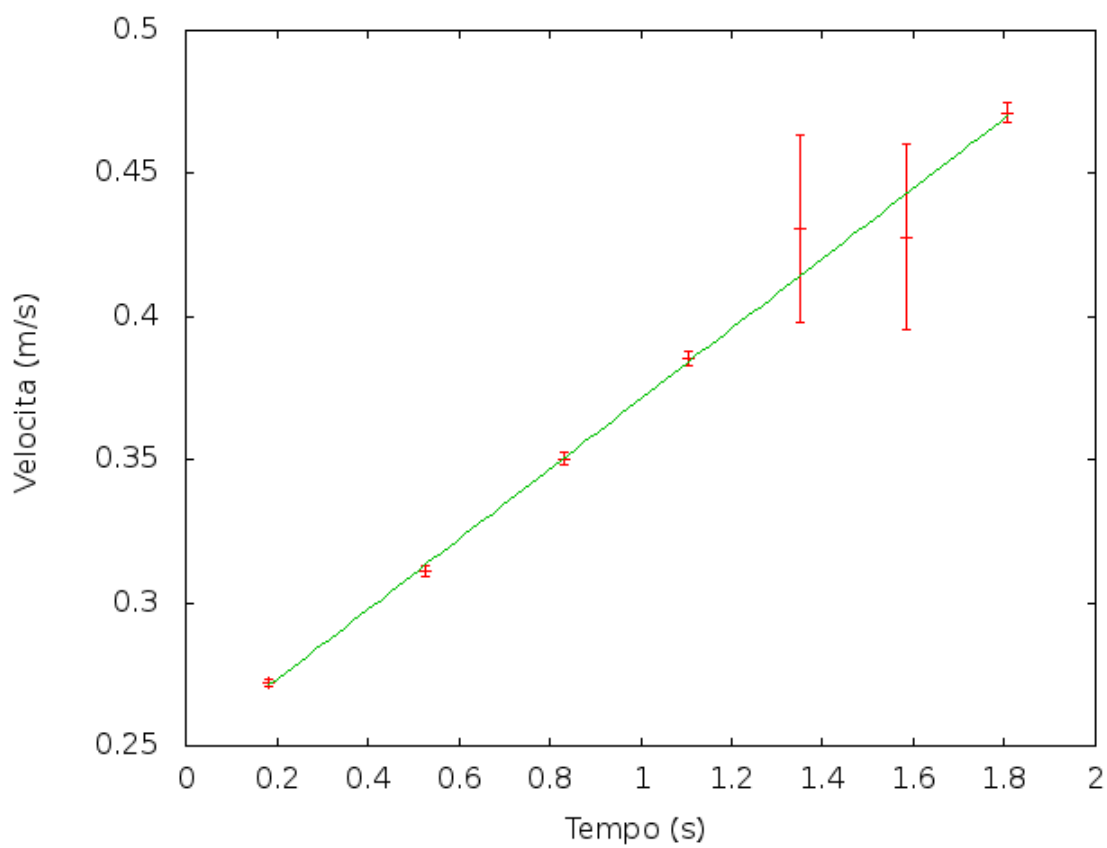
7.2 Inclinazione 30', senza peso



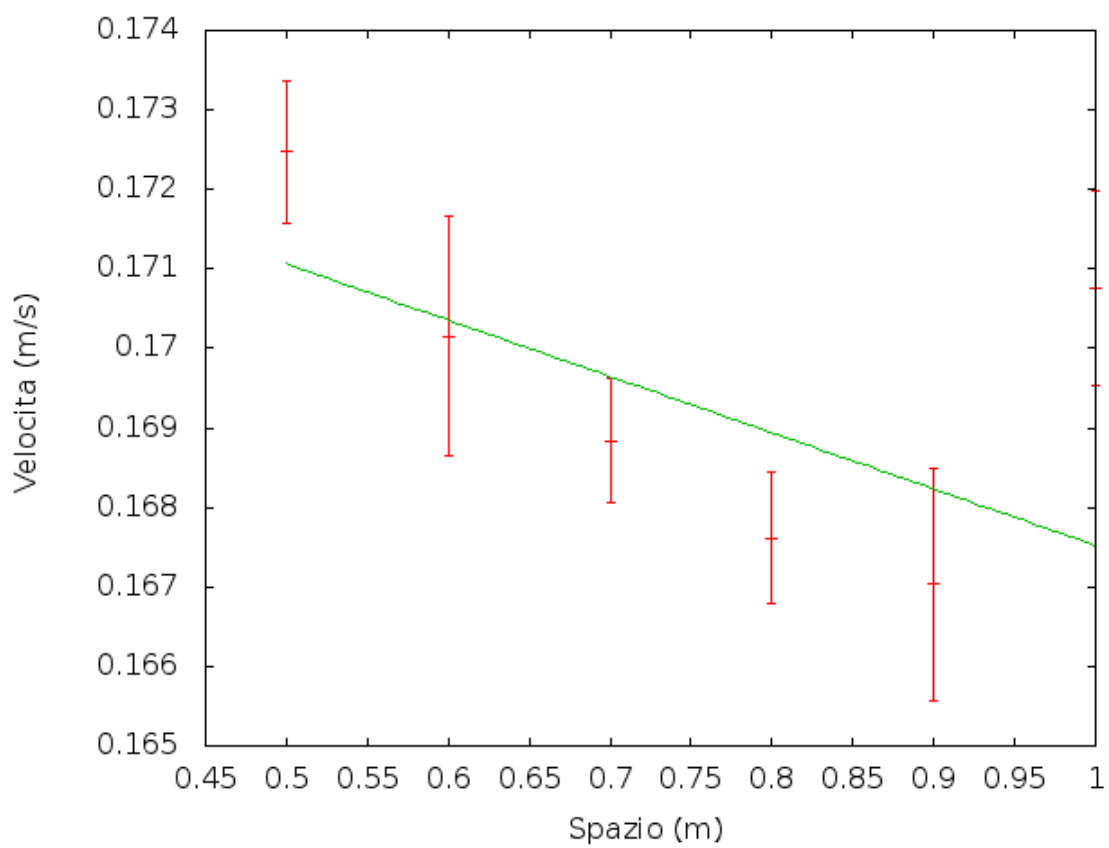
7.3 Inclinazione 45', senza peso



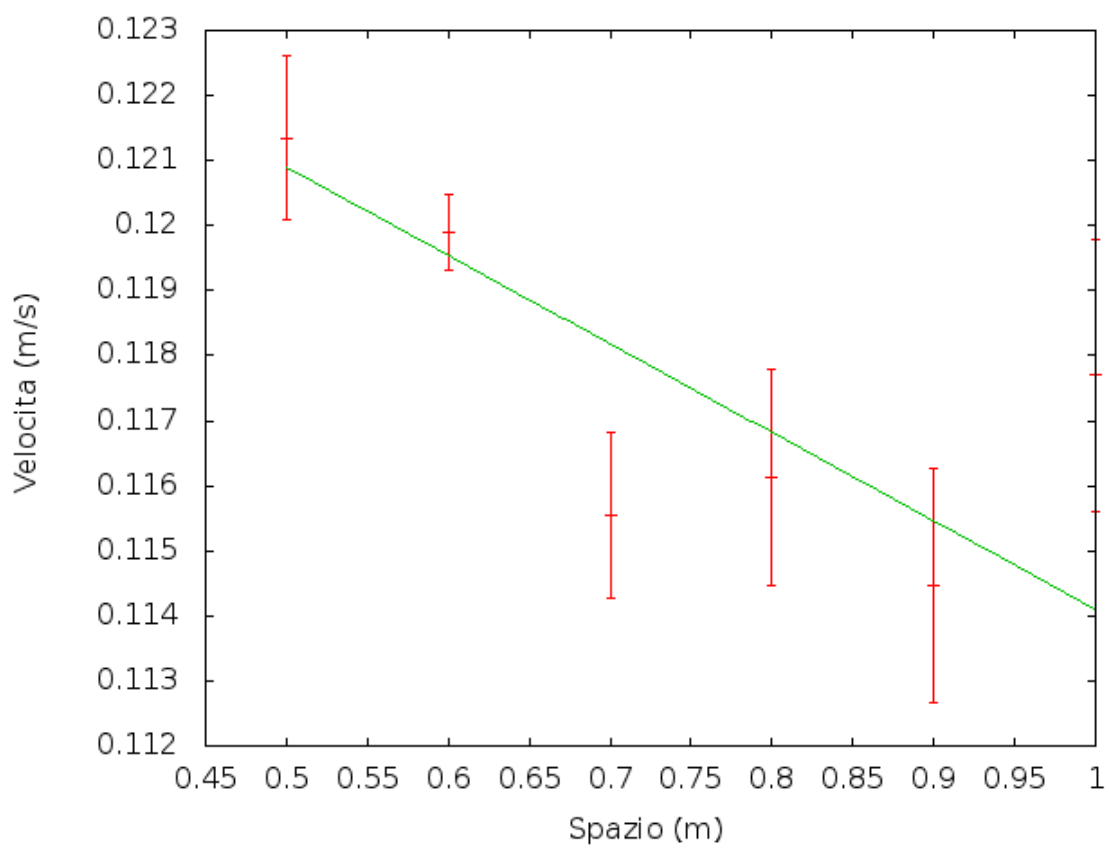
7.4 Inclinazione 45', con peso



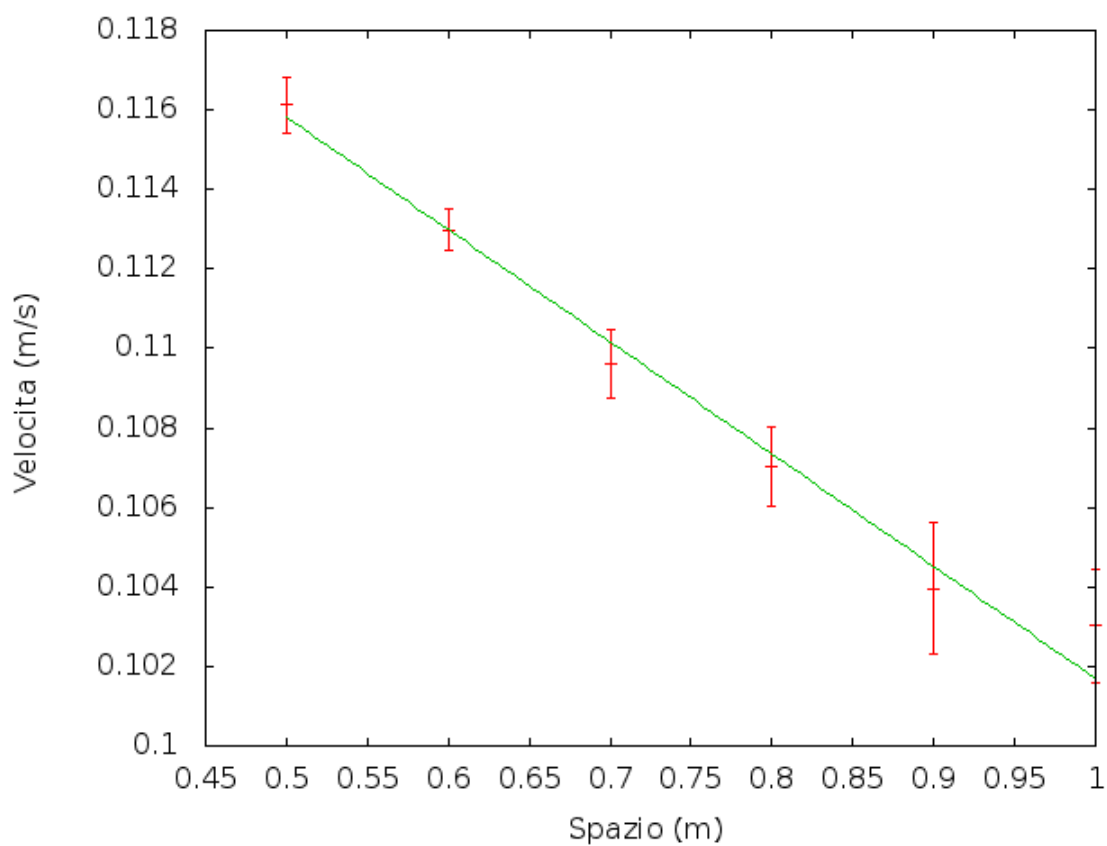
7.5 Inclinazione 0', senza peso, senza spessore



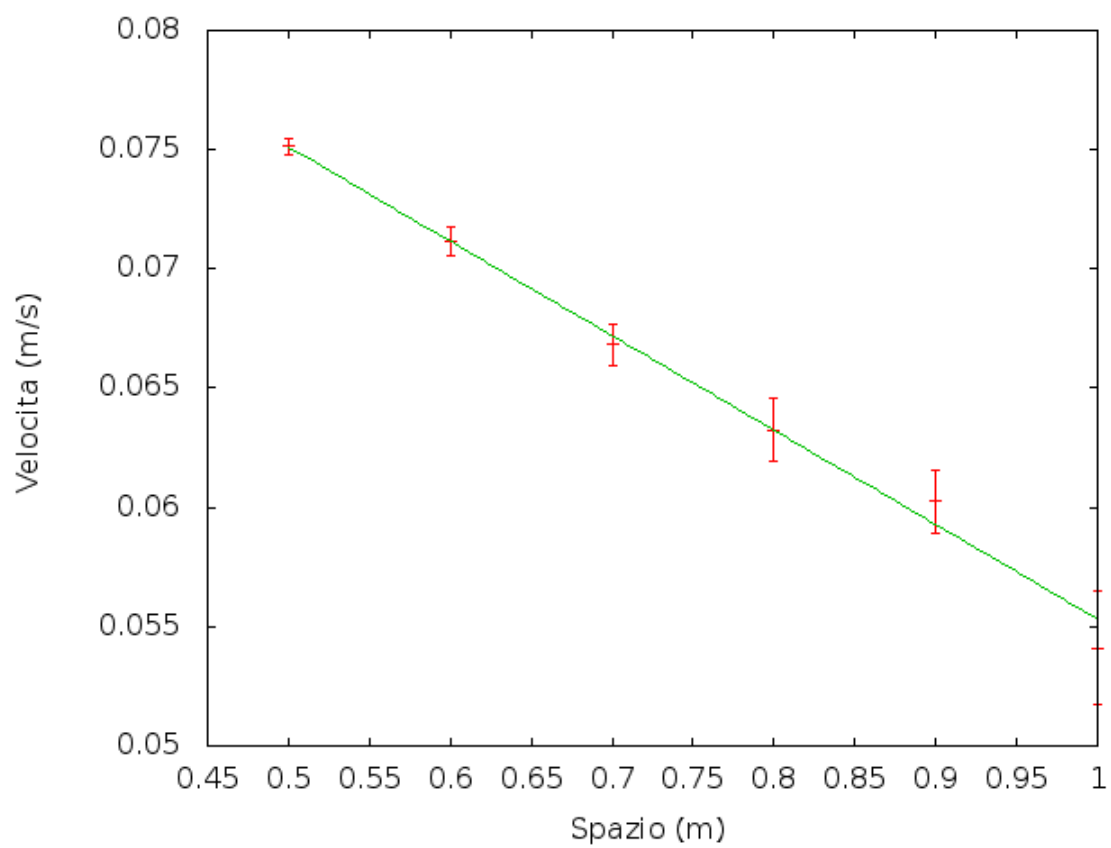
7.6 Inclinazione 0', senza peso, con spessore



7.7 Inclinazione 0', con peso, senza spessore



7.8 Inclinazione 0', con peso, con spessore



8 Codice

Riportiamo in seguito il programma utilizzato per l'elaborazione dei dati. Fondamentalmente abbiamo creato una classe template (una gerarchia, a dire la verità, ma non abbiamo usato le specializzazioni) che rappresenta una variabile statistica: contiene la media, la varianza del campione e quella della popolazione, la deviazione standard del campione e quella della popolazione, il massimo, il minimo e l'errore della media. La classe base, (la sola usata qui) legge i dati solo all'inizio nel costruttore, poi astrae da essi e grazie all'overloading degli operatori di addizione, sottrazione, moltiplicazione e divisione, risulta facile ed efficiente da manipolare: espressioni complesse di variabili statistiche, dato che i vari errori vengono propagati dietro le quinte automaticamente, possono essere espresse chiaramente e concisamente nel codice, cosa che facilita il controllo e velocizza la scrittura.

```
//=====
// Name      : Misure.cpp
// Author     : Francesco Forcher
// Version    : 1.5
// Description : Programma per analizzare i dati sul pendolo raccolti in laboratorio
//=====

////////////////////////////////////
//Librerie
#include <iostream>//cin e cout
#include <fstream>//FileStream
#include <exception>//Eccezioni
#include <string>
#include <cstdlib>//system(clear)
#include <algorithm>//Sort?
#include <sstream>//StringStream

////////////////////////////////////
//le mie classi

//In file "VarStat.h"
#pragma once//L'equivalente delle Include guards

#include <cmath>
#include <vector>
#include <algorithm>
```

```
#include <memory>
#include <limits>
#include <cmath>

#ifdef _MIO_DEBUG_

#include <iostream>//Per cerr

//Stampa il nome della variabile
#define VNAME(x) #x
#define VDUMP(x) std::clog << #x << " " << x << std::endl

#endif

//Il mio namespace
namespace mions {
//Classi per l'analisi dei dati statistici
namespace dataAnalisi {
using std::vector;

////////////////////
// //
// VERSIONE 1.3 //
// //
////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*
 * Forward declarations
 */
template <typename> class VarStat;//Forward declaration da usare nella funzione operator<<

template <typename T> const VarStat<T> operator*(const double& , const VarStat<T> );
template <typename T> const VarStat<T> operator*(const VarStat<T>, const double& );
```



```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Versione
template <typename U>
std::ostream& operator <<(std::ostream& os, const VarStat<U>& rhs) {
using namespace std;

//Eclipse dà problemi con endl, modifichiamolo temporaneamente
#define endl "\n"
os << "Numero dati:                " << rhs.getNumeroDatiEffettivo() << endl;
os << "Media:                        " << rhs.getMedia() << endl;
//cout << "Mediana:                " << rhs.getMediana() << endl;
os << "Varianza del campione:         " << rhs.getVarianzaCampione() << endl;
os << "Deviazione standard campione: " << rhs.getDeviazioneStandardCamp() << endl;
os << "Varianza della popolazione:    " << rhs.getVarianzaPopolazione() << endl;
os << "Deviazione standard popolazione: " << rhs.getDeviazioneStandardPop() << endl;
os << "Errore della media:           " << rhs.getErroreMedia() << endl;
os << "Massimo:                     " << rhs.getMax() << endl;
os << "Minimo:                      " << rhs.getMin() << endl;
#undef endl

return os;

}

//Moltiplicazione a destra per uno scalare
template <typename U>
inline const VarStat<U> operator*(const VarStat<U> lhs, const double& rhs) {
VarStat<U> result = lhs; // Copia il primo oggetto
result *= rhs;          // Aggiungici dentro l'altro
return result;          // Ritorna il risultato
}

//Moltiplicazione a sinistra per uno scalare
template <typename U>
inline const VarStat<U> operator*(const double& lhs, const VarStat<U> rhs) {
return (rhs*lhs);
}

```

```
//Moltiplicazione a sinistra per uno scalare
```

////////////////////////////////////

```
// Classe per l'analisi di UNA variabile statistica offline, cioè avendo accesso a tutti i dati
```

```
// 0 anche, che "rappresenta" una variabile statistica
```

```
template <class T>
```

```
class VarStat {
```

```
public:
```

```
//vector<T> vectDati;
```

```
//Funzioni overloaded
```

```
friend std::ostream& operator<<<T>(std::ostream& , const VarStat<T>& );
```

```
friend const VarStat<T> operator* $\langle$ T $\rangle$ (const VarStat<T> , const double& );
```

```
friend const VarStat<T> operator*<T>(const double& , const VarStat<T> );
```

//

```
VarStat(T valore) {
```

```
iNumero_dati = 1;
```

```
dMedia = (double)valore;
```

```
dDeviazioneStandardCamp = 0;
```

```
dDeviazioneStandardPop = 0;
```

```
dVarianzaCampione = 0;
```

```
dVarianzaPopolazione = 0;
```

```
dMax = (double)valore;
```

```
dMin = (double)valore;
```

```
dErroreMedia = 0;
```

}

```
VarStat(T valore, double DevStdPop, int numDati = 100000) {
```

```
iNumero_dati = numDati;
```

```
dMedia = (double)valore;
```

```
dDeviazioneStandardPop = DevStdPop;
```

```
dDeviazioneStandardCamp = DevStdPop * sqrt(double(numDati-1)/double(numDati));
```

```
dVarianzaCampione = dDeviazioneStandardCamp*dDeviazioneStandardCamp;
```

```
dVarianzaPopolazione = DevStdPop*DevStdPop;
```

```
dMax = (double)valore + DevStdPop;
```

```
dMin = (double)valore - DevStdPop;
```

```

dErroreMedia = DevStdPop / sqrt(numDati);
}

//Costruttore
VarStat(const vector<T>& aDati, bool eliminaTreSigma = true) {
//aDati = {1,2,3}; //La classe ha una copia del vector! Non dei dati! Copiare un vector non è t
int numDatiIniziale = aDati.size();
vector<int> ListaDatifuori3Sigma; //0.003 = 100% - 99.7% = percentuale attesa di fuori sigma,

//Se il vettore è vuoto la random variable è 0 +- 0 Buona idea?
if (numDatiIniziale == 0) {

#ifdef _MIO_DEBUG_
std::clog << "Vettore vuoto, metto la variabile a zero+-zero";
#endif

iNumero_dati = 0;
dMedia = 0;
dDeviazioneStandardCamp = 0;
dDeviazioneStandardPop = 0;
dVarianzaCampione = 0;
dVarianzaPopolazione = 0;
dMax = 0; //Oppure +INFINITY
dMin = 0; //0 -INFINITY
dErroreMedia = 0;
return;
}

dMedia=(double)aDati[0];
dMax=(double)aDati[0];
dMin=(double)aDati[0];

for(int i=0; i < numDatiIniziale; i++){
//Media
dMedia=(i*dMedia+(double)aDati[i])/(i+1);

//Massimo e minimo (ottimizzabile?)

```

```

dMax = (aData[i] > dMax) ? aData[i] : dMax;
dMin = (aData[i] < dMin) ? aData[i] : dMin;
}

dVarianzaCampione=pow(((double)aData[0]-dMedia),2);
for(int i=0; i < numDatiIniziale; i++){
//Varianza
dVarianzaCampione=(i*dVarianzaCampione+pow(((double)aData[i]-dMedia),2)) /
(i+1);
}

dDeviazioneStandardCamp = sqrt(dVarianzaCampione);

//se sigma2c=S/N e sigma2p=S/(N-1), allora, sostituendo S e risolvendo, sigma2p=sigma2c*N/(N-1)
dVarianzaPopolazione = dVarianzaCampione*double(numDatiIniziale)/(double(numDatiIniziale)-1);
iNumero_dati = aData.size();
////////////////////////////////////
//Se eliminaTreSigma è true, rifai i conti togliendo i dati inaccettabili
int numCancellazioni = 0;
if (eliminaTreSigma){
std::clog << "Elimino i dati oltre 3 sigma...\n" ;
/* pData è un tipo vector<double>::iterator, e si comporta come un puntatore a un elemento del
* Sarebbe più leggibile scrivere "auto pData = vectDati.begin();", ma per chiarezza mettiamo
*
* typename è richiesto perchè se qualcuno scrivesse "T::iterator * iter;" e se per esempio T
* sarebbe una moltiplicazione (stupido c++), quindi dobbiamo specificare che intendiamo un ti
* http://pages.cs.wisc.edu/~driscoll/typename.html#real_reason
*
* Non incrementiamo l'iteratore (pData++) nell'istruzione for, invece lo assegnamo nel ciclo
*/
int i = 0;
for (typename vector<T>::const_iterator pData = aData.begin();
pData != aData.end();
pData++, i++)// i indica l'offset dall'inizio del vector, lo useremo dopo per verificare i dati
{
//i++;//Per mettere gli offset degli iterator, probabilmente si può mettere nel ciclo direttam
if (abs(dMedia - (*pData) ) >= 3*dDeviazioneStandardCamp) {

```

```
* Cancelliamo dal Vector i dati incrociabili. Operazione costosa perchè i dati successivi ve  
* indietro, ma è meglio un Vector di una LinkedList perchè i dati possono essere messi nella  
* erase richiede un iterator, quindi siamo "costretti" a usarlo  
*/  
std::clog << "Eliminato dato: " << *pData << "\n";  
  
ListaDatifuori3Sigma.push_back(i); //Aggiungi il dato nella posizione i alla lista degli "incroci"  
  
numCancellazioni = numCancellazioni + 1;  
}  
}  
  
std::clog << "Cancellati " << numCancellazioni << " dati\n\n";  
} //EndIf  
/////////////////////////////////////  
  
if ( !(ListaDatifuori3Sigma.empty()) ) {  
for (int k = 0; k < numCancellazioni; ++k) {  
std::cerr << ListaDatifuori3Sigma[k] << std::endl;  
}  
  
//Rifacciamo i conti  
numDatiIniziale = aDati.size();  
dMedia=(double)aDati[0];  
dMax=(double)aDati[0];  
dMin=(double)aDati[0];  
  
// j è l'indice del vettore che contiene gli indici dei dati da scartare. Ovviamente si suppone  
int j = 0;  
for(int i=0; i < numDatiIniziale; i++) {  
//Media  
//Se il dato è buono prosegui coi calcoli  
if ( i != *(ListaDatifuori3Sigma.begin() + j) ) {  
dMedia=(i*dMedia+(double)aDati[i])/(i+1);  
  
//Massimo e minimo  
dMax = (aDati[i] > dMax) ? aDati[i] : dMax;  
dMin = (aDati[i] < dMin) ? aDati[i] : dMin;
```

```
} else {  
//Il dato è da scartare, prosegui con quello successivo (e avanza al prossimo numero nella lista)  
j++;  
}  
  
// Di nuovo j  
j=0;  
dVarianzaCampione=pow(((double)aDati[0]-dMedia),2);  
for(int i=0; i < numDatiIniziale; i++) {  
//Varianza  
if (i != *(ListaDatifuori3Sigma.begin() + j)) {  
dVarianzaCampione=(i*dVarianzaCampione+pow((((double)aDati[i]-dMedia),2)) /  
(i+1));  
} else {  
j++;  
}  
}  
  
dDeviazioneStandardCamp = sqrt(dVarianzaCampione);  
  
//se sigma2c=S/N e sigma2p=S/(N-1), allora, sostituendo S e risolvendo, sigma2p=sigma2c*N/(N-1)  
dVarianzaPopolazione = dVarianzaCampione*double(numDatiIniziale)/(double(numDatiIniziale)-1);  
  
} //EndIf del ricalcolo  
/////////////////////////////////////  
  
//Deviazione standard popolazione  
dDeviazioneStandardPop=sqrt(dVarianzaPopolazione);  
iNumero_dati = numDatiIniziale - ListaDatifuori3Sigma.size();  
dErroreMedio = dDeviazioneStandardPop / sqrt(iNumero_dati);  
  
} //Fine costruttore  
  
// Distruttore  
virtual ~VarStat() = default; //Virtual perchè devono ereditare da questa. Lecito il default? E  
  
//Getters
```

```

inline double getMedia() const {return dMedia;};
//Scarto Quadratico Medio (N)
inline double getDeviazioneStandardCamp() const {return dDeviazioneStandardCamp;};
//Errore Quadrato Medio N-1
inline double getDeviazioneStandardPop() const {return dDeviazioneStandardPop;};
//Su Excel sono invertite, cioè per varianza del campione io intendo la varianza propria dei d
inline double getVarianzaCampione() const {return dVarianzaCampione;};
//Su Excel sono invertite, cioè per varianza popolazione io considero implicitamente i dati co
//la varianzaPopolazione è calcolata fratto N-1
inline double getVarianzaPopolazione() const {return dVarianzaPopolazione;};
//double getMediana() ordina i dati come side effect
//Tolta
inline double getMax() const {return dMax;};
inline double getMin() const {return dMin;};
// Errore della media
inline double getErroreMedia() const {return dErroreMedia;};
long getNumeroDatiEffettivo() const {return iNumero_dati;};
//Range della variabile
inline long getRange() const {return dMax - dMin;};
//double getModa() const {return dModa;};

//Operatori
//Somma una variabile statistica a un'altra e memorizzala nella prima. Vedi commento su -=, so
inline VarStat<T>& operator+=(const VarStat<T>& rhs) {
using std::abs;
//Obiettivo: dare gli stessi risultati come se avessi sommato i dati di due insiemi (i dati du
//iNumero_dati = iNumero_dati; Non sto unendo gli insiemi di dati, ma sommando i singoli eleme
dMedia += rhs.getMedia(); //Somma le medie delle due variabili
dVarianzaCampione = rhs.getVarianzaCampione() + getVarianzaCampione(); //Propagazione dell'erro
dVarianzaPopolazione = rhs.getVarianzaPopolazione() + getVarianzaPopolazione();

//La nuova varianza permette di calcolare direttamente la nuova std
dDeviazioneStandardCamp = sqrt(getVarianzaCampione());
dDeviazioneStandardPop = sqrt(getVarianzaPopolazione());

// Il massimo della somma è la somma dei due massimi
// Worst-case max? Non ben definito, ma se ho due set di dati, il massimo (tra tutte le possib

```

[illegible]


```

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//Operatori
//moltiplica una variabile statistica a un'altra e memorizzala nella prima. Vedi commento su -
inline VarStat<T>& operator*=(const VarStat<T>& rhs) {
using std::abs;
//Obiettivo: dare gli stessi risultati come se avessi moltiplicato i dati di due insiemi (i da
//iNumero_dati = iNumero_dati; Non sto unendo gli insiemi di dati, ma moltiplicando i singoli
double tMedia = getMedia(); //Salvo la media
dMedia = rhs.getMedia() * getMedia(); //Moltiplica le medie delle due variabili
dVarianzaCampione = getMedia()*getMedia()*(getVarianzaCampione() / (tMedia*tMedia) + rhs.getVa
dVarianzaPopolazione = getMedia()*getMedia()*(getVarianzaPopolazione() / (tMedia*tMedia) + rhs

//La nuova varianza permette di calcolare direttamente la nuova std
dDeviazioneStandardCamp = sqrt(getVarianzaCampione());
dDeviazioneStandardPop = sqrt(getVarianzaPopolazione());

// Il massimo della somma è la somma dei due massimi
// Worst-case max? Non ben definito, ma se ho due set di dati, il massimo (tra tutte le possib
// TODO: Casi non maggiori di zero
// TODO: Casi non maggiori di zero
dMax = (getMedia() > 0) ? (getMax() * rhs.getMax()) : ( INFINITY );

//Idem per il minimo, il minimo "minore" è la somma dei minimi
dMin = (getMedia() > 0) ? (getMin() * rhs.getMin()) : ( -INFINITY );

dErroreMedia = getDeviazioneStandardPop() / sqrt(getNumeroDatiEffettivo()); //Propagato come un
return *this; //Idiozia ma dicono che serva
}

inline VarStat<T>& operator/=(const VarStat<T>& rhs) {
using std::abs;
//Obiettivo: dare gli stessi risultati come se avessi moltiplicato i dati di due insiemi (i da
//iNumero_dati = iNumero_dati; Non sto unendo gli insiemi di dati, ma moltiplicando i singoli
double tMedia = getMedia(); //Salvo la media
dMedia = getMedia() / rhs.getMedia(); //Moltiplica le medie delle due variabili
dVarianzaCampione = getMedia() * getMedia()*(getVarianzaCampione() / (tMedia*tMedia) + rhs.get
dVarianzaPopolazione = getMedia() * getMedia()*(getVarianzaPopolazione() / (tMedia*tMedia) +rh

```

```

//La nuova varianza permette di calcolare direttamente la nuova std
dDeviazioneStandardCamp = sqrt(getVarianzaCampione());
dDeviazioneStandardPop = sqrt(getVarianzaPopolazione());

// Il massimo della somma è la somma dei due massimi
// Worst-case max? Non ben definito, ma se ho due set di dati, il massimo (tra tutte le possibili somme)
// TODO: Casi non maggiori di zero
dMax = (getMedia() > 0) ? (getMax() / rhs.getMin()) : ( INFINITY );

//Idem per il minimo, il minimo "minore" è la somma dei minimi
dMin = (getMedia() > 0) ? (getMin() / rhs.getMax()) : ( -INFINITY );

dErroreMedia = getDeviazioneStandardPop() / sqrt(getNumeroDatiEffettivo());
return *this; //Idiozia ma dicono che serva
}

```

```

//Moltiplicazione per scalare, compound assignment. v *= d è come v.operator*=(d), quindi le f
inline VarStat<T>& operator*=(const double& rhs) {
using std::abs;
//Obiettivo: dare gli stessi risultati come se avessi preso tutti i dati e, moltiplicato ciasco
dMedia = rhs * getMedia(); //Moltiplica la media della VarStat a sinistra del simbolo per il do
dVarianzaCampione = rhs * rhs * getVarianzaCampione();
dVarianzaPopolazione = rhs * rhs * getVarianzaPopolazione();
dDeviazioneStandardCamp = abs(rhs) * getDeviazioneStandardCamp();
dDeviazioneStandardPop = abs(rhs) * getDeviazioneStandardPop();

//Se moltiplico per uno scalare negativo, il minimo nei positivi diventa il massimo nei negati
T tMax = dMax; //Massimo temporaneo, perchè quando calcoliamo dMin è già stato modificato dmax
T tMin = dMin;

```

```

dMax = (rhs >= 0 ? rhs * tMax : rhs * tMin);
dMin = (rhs >= 0 ? rhs * tMin : rhs * tMax);

dErroreMedia = abs(rhs)*getErroreMedia();
return *this; //Idiozia ma dicono che serva
}

//Somma di due VarStat
//Trucchetto per riutilizzare il lavoro svolto con +=
const VarStat<T> operator+(const VarStat<T> &other) const {
VarStat<T> result = *this; // Copia il primo oggetto
result += other;           // Aggiungici dentro l'altro
return result;             // Ritorna il risultato
}

//Sottrazione di due VarStat
//Trucchetto per riutilizzare il lavoro svolto con -=
const VarStat<T> operator-(const VarStat<T>& other) const {
VarStat<T> result = *this; // Copia il primo oggetto
result -= other;           // Aggiungici dentro l'altro
return result;             // Ritorna il risultato
}

const VarStat<T> operator*(const VarStat<T>& other) const {
VarStat<T> result = *this; // Copia il primo oggetto
result *= other;           // Aggiungici dentro l'altro
return result;             // Ritorna il risultato
}

const VarStat<T> operator/(const VarStat<T>& other) const {
VarStat<T> result = *this; // Copia il primo oggetto
result /= other;           // Aggiungici dentro l'altro
return result;             // Ritorna il risultato
}

//Moltiplicazione per uno scalare

```

```
//Trucchetto per riutilizzare il lavoro svolto con *= double,
// const VarStat<T> operator*(const double rhs) const {
// VarStat<T> result = *this; // Copia il primo oggetto
// result *= rhs;           // Aggiungici dentro l'altro
// return result;           // Ritorna il risultato
// }
```

```
private:
```

```
double dMedia = -INFINITY;
double dDeviazioneStandardCamp = -INFINITY;
double dDeviazioneStandardPop = -INFINITY;
double dVarianzaCampione = -INFINITY;
double dVarianzaPopolazione = -INFINITY;
//double dMediana = -INFINITY;
double dMax = -INFINITY;
double dMin = -INFINITY;
double dErroreMedia = -INFINITY;
int iNumero_dati = 0;
//double dModa=0;

};
```

```
}//Fine DataAnalisi
```

```
}//Fine del mio namespace
```

```
//In file "SortingVarStat.h"
/*
 * SortingVarStat.h
 *
 * Created on: Feb 19, 2014
 * Author: francesco
```

```

*/

#ifndef SORTINGVARSTAT_H_
#define SORTINGVARSTAT_H_

#include "VarStat.h"

////////////////////
//  //
//   VERSIONE 0.1  //
//  //
////////////////////

namespace mions {
namespace dataAnalisi {

template <typename> class Sorting_VarStat;

//Notare come non sia const Sorting_VarStat<U>&, poichè la mediana non viene calcolata finchè
template <typename U>
std::ostream& operator <<(std::ostream& os, Sorting_VarStat<U>& rhs) {
using namespace std;

//Eclipse dà problemi con endl, modifichiamolo temporaneamente
#define endl "\n"
os << "Numero dati:           " << rhs.getNumeroDatiEffettivo() << endl;
os << "Media:                   " << rhs.getMedia() << endl;
os << "Mediana:                 " << rhs.getMediana() << endl;
os << "Varianza del campione:   " << rhs.getVarianzaCampione() << endl;
os << "Deviazione standard campione: " << rhs.getDeviazioneStandardCamp() << endl;
os << "Varianza della popolazione: " << rhs.getVarianzaPopolazione() << endl;
os << "Deviazione standard popolazione: " << rhs.getDeviazioneStandardPop() << endl;
os << "Errore della media:      " << rhs.getErroreMedia() << endl;
os << "Massimo:                 " << rhs.getMax() << endl;
os << "Minimo:                  " << rhs.getMin() << endl;
#undef endl

```

```

return os;
};

template <class T>
class Sorting_VarStat: public mions::dataAnalisi::VarStat<T> {
public:
friend std::ostream& operator<<<T>(std::ostream& os, Sorting_VarStat<T>& rhs);

vector<T> vectDati;
Sorting_VarStat(const vector<T>&& aDati, bool eliminaTreSigma = true) : mions::dataAnalisi::Va
vectDati = aDati;//Copia i dati, così puoi riordinarli in santa pace. In teoria move-assignm
};

virtual ~Sorting_VarStat() throw() {};

double getMediana() {
if (dMediana != -INFINITY) {
return dMediana;
}
else
{
ordinaDati();
int numDati = this->getNumeroDatiEffettivo();
if (numDati % 2 == 1) {
return dMediana = (double)vectDati[(numDati-1)/2];
} else {
return dMediana = (double)(vectDati[numDati/2-1]+vectDati[numDati/2])/2;
}
}
}

////////////////////////////////////
private:

bool DatiOrdinati = false;
double dMediana = -INFINITY;//Un valore che non dovrebbe assumere mai...

```

```

inline void ordinaDati() {
if (!DatiOrdinati) {
std::sort(vectDati.begin(),vectDati.end());
DatiOrdinati = true;
}
}
};

} /* namespace dataAnalisi */
} /* namespace mions */

#endif /* SORTINGVARSTAT_H_ */

#define VERSIONE 1.5

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Prototipi

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Il primo argomento è la cartella dei dati
int main(int numParam, char* args[]) {

using namespace std;

system("clear");
cout << "\n";
cout << "Programma per analizzare i dati della guidovia, versione: " << VERSIONE << endl;
//Ricordarsi che con 0 gradi l'intervallo era di 20!
const int NUM_FILE = 7;// 7 file (7 intervalli) per 15, 30, 45 gradi
const int NUM_FILE_OGRADI = 6; // file per 60,70,80,90,100,110
const int NUM_DATIPERFILE = 5;// 5 dati in ogni file
const int ANGOLI_NUM = 3;//15, 30, 45 0 e 45 li facciamo a parte
//const auto INTERVALLO = VarStat<double>(0.1, 0.001 / sqrt(6));//Distribuzione triangolare

```

[illegible]


```

string nf;//Nome file da aprire
//FileStream
ofstream FileMedie;//FileStream
ofstream FileRiassunto;
ofstream FileVelocita;
using namespace mions::dataAnalisi;
vector<VarStat<double> > arrayRiassunti;//Contiene le informazioni come la deviazione standard
vector<VarStat<double> > arrayTempi;

arrayRiassunti.reserve(NUM_FILE);
string nomeoutputfile;
string nomefilemedie;
string nomefilevelocita;
string nomeoutputvelocita;//Per tenere temporaneamente la variabile ed evitare il self assignm

/* Vari casi:
 * 1. n  peso n  alluminio
 */
string tipodati;
nomeoutputfile = "./Risultati/normale_dati.txt";
//tipodati = "d";
nomefilemedie = "arrayTempi_PrimaVolta_normale.txt";
nomefilevelocita = "velocita_PV_normale.txt";//PV = prima volta

FileRiassunto.open(nomeoutputfile.c_str());

for (int angoli = 1; angoli <= ANGOLI_NUM; angoli++) {
//TODO aggiunto zero all'inizio
arrayTempi.emplace_back(0.0);

for (int intervallo = 1; intervallo <= NUM_FILE; intervallo++) {

stringstream ss;
ifstream FileInputDati;//File di input
ss << "./DatiFormattati/DatiStandardizzati/";
ss << "d";//Tipo dei dati contenuti nel file: d, cd
ss << intervallo*10 + 40;

```

```

ss << "_";
ss << angoli*15;
ss >> nf;
//ss.flush();

clog << nf << endl;
FileInputDati.open(nf.c_str()); // Apro il file indicato nell'argomento dato via shell
if (!FileInputDati.is_open())
throw "Errore: file non aperto";

vector<double> tempVect(5); // Vector che contiene i dati di un file solo, da cui ricavare il t
tempVect.reserve(NUM_DATIPERFILE);

clog << tempVect.size() << endl;
for (unsigned int i = 0; i < tempVect.size(); i++) {
FileInputDati >> tempVect[i];
clog << "Pos " << i << ": " << tempVect[i] << endl;
}
clog << "Dati letti. Analizzo..." << endl << endl;

arrayRiassunti.emplace_back(tempVect, true); // Forwarda gli argomenti a un oggetto costruito DI

cout << endl;
cout << "Nome file: " << nf << endl;
cout << arrayRiassunti.back();

FileRiassunto << endl;
FileRiassunto << "Nome file: " << nf << endl;
FileRiassunto << arrayRiassunti.back() << endl;

arrayTempi.emplace_back(arrayRiassunti.back());

ss.clear();
FileInputDati.close();
} // Intervalli

```

```

switch (angoli) {
case 1:
//Ricicliamo nomeoutputfile per indicare i file di uscita
nomeoutputfile = string("./Risultati/MetaDati/15/") + nomefilemedie;
nomeoutputvelocita = string("./Risultati/MetaDati/15/") + nomefilevelocita;
FileMedie.open(nomeoutputfile.c_str());
FileVelocita.open(nomeoutputvelocita.c_str());
//const auto intervallo = VarStat<double>(0.1, 0.1 / sqrt(6));
for (int i = 0; i < NUM_FILE; i++)
{
FileMedie << ((arrayTempi[i+1]+arrayTempi[i])/2).getMedia() << endl;//sette medie di cinque te
FileVelocita << (INTERVALLO_PV / (arrayTempi[i+1] - arrayTempi[i]) ).getMedia() << " "
<< (INTERVALLO_PV / (arrayTempi[i+1] - arrayTempi[i]) ).getDeviazioneStandardPop() << endl;//1
}
FileMedie.close();
FileVelocita.close();
break;
case 2:
//Ricicliamo nomeoutputfile per indicare i file di uscita
nomeoutputfile = string("./Risultati/MetaDati/30/") + nomefilemedie;
nomeoutputvelocita = string("./Risultati/MetaDati/30/") + nomefilevelocita;
FileMedie.open(nomeoutputfile.c_str());
FileVelocita.open(nomeoutputvelocita.c_str());
for (int i = 0; i < NUM_FILE; i++)
{
FileMedie << ( (arrayTempi[i+1] + arrayTempi[i]) / 2 ).getMedia() << endl;//sette medie di cin
FileVelocita << (INTERVALLO_PV / (arrayTempi[i+1] - arrayTempi[i]) ).getMedia() << " "
<< (INTERVALLO_PV / (arrayTempi[i+1] - arrayTempi[i]) ).getDeviazioneStandardPop() << endl;//1
}
FileMedie.close();
FileVelocita.close();
break;
case 3:
//Ricicliamo nomeoutputfile per indicare i file di uscita
nomeoutputfile = string("./Risultati/MetaDati/45/") + nomefilemedie;
nomeoutputvelocita = string("./Risultati/MetaDati/45/") + nomefilevelocita;

```

```

FileMedie.open(nomeoutputfile.c_str());
FileVelocita.open(nomeoutputvelocita.c_str());
for (int i = 0; i < NUM_FILE; i++)
{
FileMedie << ((arrayTempi[i+1]+arrayTempi[i])/2).getMedia() << endl;//sette medie di cinque te
FileVelocita << (INTERVALLO_PV / (arrayTempi[i+1] - arrayTempi[i]) ).getMedia() << " "
<< (INTERVALLO_PV / (arrayTempi[i+1] - arrayTempi[i]) ).getDeviazioneStandardPop() << endl;//1
}
FileMedie.close();
FileVelocita.close();
break;
default:
throw "Errore: numero casi angoli sbagliato";
break;
};//switch
arrayTempi.clear();
};//Angoli

FileRiassunto.close();
arrayRiassunti.clear();
    };//Blocco prima esperienza senza peso
////////////////////////////////////

//Prima esperienza, guidovia inclinata a 45 con peso
////////////////////////////////////
/*
* Vari casi:
* 1. n  peso n  alluminio

```

```

* 2. peso
*/
//for (int varicasi = 1; varicasi <= 2; varicasi++)
{

string nf; // Nome file
ofstream FileMedie; // File con le medie dei tempi
ofstream FileRiassunto; // File con le informazioni sulle cinque di dati
ofstream FileVelocita;
using namespace mions::dataAnalisi;
vector<VarStat<double> > arrayRiassunti;//Contiene le informazioni come la deviazione standard
vector<VarStat<double> > arrayTempi;
//TODO aggiunto zero all'inizio
arrayTempi.emplace_back(0.0);

arrayRiassunti.reserve(NUM_FILE);
string nomeoutputfile;
string nomefilemedie;
string nomefilevelocita;
string nomeoutputvelocita;

string tipodati;
nomeoutputfile = "./Risultati/peso_dati.txt";
tipodati = "cd";
nomefilemedie = "arrayTempi_PrimaVolta_peso.txt";
nomefilevelocita = "velocita_PV_peso.txt";

//string nomeoutputfile = "./Risultati/nopeso_dati.txt";
FileRiassunto.open(nomeoutputfile.c_str());
for (int intervallo = 1; intervallo <= NUM_FILE; intervallo++) {

stringstream ss;
ifstream FileInputDati; //File di input
ss << "./DatiFormattati/DatiStandardizzati/";
ss << tipodati; //Tipo dei dati contenuti nel file: d, cd

```

```
ss << intervallo * 10 + 40;
ss << "_";
ss << "45";
ss >> nf;
//ss.flush();

clog << nf << endl;
FileInputDati.open(nf.c_str()); //Apro il file indicato nell'argomento dato via shell
if (!FileInputDati.is_open())
throw "Errore: file non aperto";

vector<double> tempVect(5); // Vector che contiene i dati di un file solo, da cui ricavare il
tempVect.reserve(NUM_DATIPERFILE);

clog << tempVect.size() << endl;
for (unsigned int i = 0; i < tempVect.size(); i++) {
FileInputDati >> tempVect[i];
clog << "Pos " << i << ": " << tempVect[i] << endl;
}
clog << "Dati letti. Analizzo..." << endl << endl;

arrayRiassunti.emplace_back(tempVect, true); // Forwarda gli argomenti a un oggetto costruito D

cout << endl;
cout << "Nome file: " << nf << endl;
cout << arrayRiassunti.back();

FileRiassunto << endl;
FileRiassunto << "Nome file: " << nf << endl;
FileRiassunto << arrayRiassunti.back() << endl;

arrayTempi.emplace_back(arrayRiassunti.back());

ss.clear();
FileInputDati.close();
} //Intervalli
```

[illegible]

```
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// Seconda giornata %
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
//Seconda esperienza, 0 gradi con alluminio e peso
```

```
/* Vari casi:
```

```
* 1. nè peso nè alluminio (file "d...")
* 2. peso (file "cd...")
* 4. alluminio
* 3. peso e alluminio ("cad...")
*/
```

```
for (int varicasi = 1; varicasi <= 4; varicasi++)
```

```
{
```

```
string nf; //Nome file da aprire
```

```
ofstream FileMedie; //File per le medie
```

```
ofstream FileRiassunto;
```

```
ofstream FileVelocita;
```

```
using namespace mions::dataAnalisi;
```

```
vector<VarStat<double> > arrayRiassunti;//Contiene le informazioni come la deviazione standard
```

```
vector<VarStat<double> > arrayTempi;//Niente zero perchè non devo dividere per la differenza
```

```
arrayRiassunti.reserve(NUM_FILE_OGRADI);
```

```
string nomeoutputfile;
```

```
string nomefilemedie;
```

```
string nomefilevelocita;
```

```
string nomeoutputvelocita;
```

```
const auto INTERVALLO_SV = VarStat<double>(0.2, 0.001 / sqrt(6));//Distribuzione triangolare
```

```
/* Vari casi:
```

```
* 1. nè peso nè alluminio (file "d...")
* 2. peso (file "cd...")
```



```

* 4. alluminio
* 3. peso e alluminio ("cad...")
*/
string tipodati;
switch (varicasi) {
case 1: // normale
nomeoutputfile = "./Risultati/normale_Ogradi_dati.txt";
tipodati = "d";
nomefilemedie = "arrayTempi_Ogradi_normale.txt";
nomefilevelocita = "velocita_Ogradi_normale.txt";
break;
case 2: // peso
nomeoutputfile = "./Risultati/peso_Ogradi_dati.txt";
tipodati = "cd";
nomefilemedie = "arrayTempi_Ogradi_peso.txt";
nomefilevelocita = "velocita_Ogradi_peso.txt";
break;
case 3: // alluminio
nomeoutputfile = "./Risultati/alluminio_Ogradi_dati.txt";
tipodati = "ad";
nomefilemedie = "arrayTempi_Ogradi_alluminio.txt";
nomefilevelocita = "velocita_Ogradi_alluminio.txt";
break;
case 4: // normale
nomeoutputfile = "./Risultati/pesoalluminio_Ogradi_dati.txt";
tipodati = "cad";
nomefilemedie = "arrayTempi_Ogradi_pesoalluminio.txt";
nomefilevelocita = "velocita_Ogradi_pesoalluminio.txt";
break;
default:
throw "Errore: Seconda esperienza: variante non nota";
break;
}

FileRiassunto.open(nomeoutputfile.c_str());

for (int intervallo = 1; intervallo <= NUM_FILE_OGRADI; intervallo++) {

```

```
stringstream ss;
ifstream FileInputDati; //File di input
ss << "./DatiFormattati/DatiStandardizzati/";
ss << tipodati; //Tipo dei dati contenuti nel file: d, cd, ad, cad
ss << intervallo * 10 + 50;
ss << "_";
ss << "0";
ss >> nf;
//ss.flush();

clog << nf << endl;
FileInputDati.open(nf.c_str()); //Apro il file indicato nell'argomento dato via shell
if (!FileInputDati.is_open())
throw "Errore: file non aperto";

vector<double> tempVect(5); // Vector che contiene i dati di un file solo, da cui ricavare il
tempVect.reserve(NUM_DATIPERFILE);

clog << tempVect.size() << endl;
for (unsigned int i = 0; i < tempVect.size(); i++) {
FileInputDati >> tempVect[i];
clog << "Pos " << i << ": " << tempVect[i] << endl;
}
clog << "Dati letti. Analizzo..." << endl << endl;

arrayRiassunti.emplace_back(tempVect, true); // Forwarda gli argomenti a un oggetto costruito D

cout << endl;
cout << "Nome file: " << nf << endl;
cout << arrayRiassunti.back();

FileRiassunto << endl;
FileRiassunto << "Nome file: " << nf << endl;
FileRiassunto << arrayRiassunti.back() << endl;

arrayTempi.emplace_back(arrayRiassunti.back());
```

```

ss.clear();
FileInputDati.close();
} //Intervalli

//Ex switch
//Ricicliamo nomeoutputfile per indicare i file di uscita
nomeoutputfile = string("./Risultati/MetaDati/0/")
+ nomefilemedie;//Nomefilemedie scelto all'inizio nello switch
nomeoutputvelocita = string("./Risultati/MetaDati/0/")
+ nomefilevelocita;//Idem per nomefilevelocita
FileMedie.open(nomeoutputfile.c_str());
FileVelocita.open(nomeoutputvelocita.c_str());
for (int i = 0; i < NUM_FILE_OGRADI; i++)
{
//Qui non c'è lo zero, quindi niente media
FileMedie << (arrayTempi[i]).getMedia() << endl;//sette medie di cinque tempi ciascuns
FileVelocita << (INTERVALLO_SV / arrayTempi[i]).getMedia() << " "
<< (INTERVALLO_SV / arrayTempi[i]).getDeviazioneStandardPop() << endl;//10 cm di intervallo/ci
}
FileMedie.close();
FileVelocita.close();

arrayTempi.clear();
FileRiassunto.close();
arrayRiassunti.clear();

}
////////////////////
////////////////////
////////////////////

```

```

////////////////////////////////////
//Analizza i dati per la gravità
typedef VarStat<double> vs;
const double G15 = 0.25*M_PI/180.0;//15 primi di grado
const double G30 = 0.50*M_PI/180.0;//15 primi di grado
const double G45 = 0.75*M_PI/180.0;//15 primi di grado

ofstream AnalisiGravita;
AnalisiGravita.open("./Risultati/Analisi_Dati/DatiGravità");

////Media tra gravità a 15, 30, 45, 45peso presi dai valori di Gnuplot
//AnalisiGravita << "Media gravità della prima giornata: (a15 + a30 + a45 + a45p)/4" << endl;
//AnalisiGravita << ((vs(0.0388864,0.001)*(1/sin(G15))).getMedia() + //15 norm
// (vs(0.0811784,0.001408)*(1/sin(G30))).getMedia() + //30 norm
// (vs(0.121616,0.001795)*(1/sin(G45))).getMedia() + //45 norm
// (vs(0.122322,0.00114)*(1/sin(G45))).getMedia() ) / //45 peso
// 4
// << endl << endl;

//Media tra gravità a 15, 30, 45, 45peso presi dai valori di Gnuplot
vs stimaGravita_orig = ((vs(0.0388864,0.001,7)*(1/sin(G15))) + //15 norm
    (vs(0.0811784,0.001503,7)*(1/sin(G30))) + //30 norm
    (vs(0.121616,0.001991,7)*(1/sin(G45))) + //45 norm
    (vs(0.122322,0.00114,7)*(1/sin(G45))) ) * //45 peso
    (0.25);

AnalisiGravita << "Dati gravità della prima giornata: (a15 + a30 + a45 + a45p)/4" << endl;
AnalisiGravita << stimaGravita_orig << endl << endl;

AnalisiGravita << "a15: " << endl << vs(0.0388864,0.001,7)*(1/sin(G15)) << endl;
AnalisiGravita << "a30: " << endl << vs(0.0811784,0.001503,7)*(1/sin(G30)) << endl;
AnalisiGravita << "a45: " << endl << vs(0.121616,0.001991,7)*(1/sin(G45)) << endl;
AnalisiGravita << "a45p: " << endl << vs(0.122322,0.00114,7)*(1/sin(G45)) << endl << endl;

```

```

//Media Coefficienti rette senza peso
vs b_np = (vs(0.00707886,0.004894,6) + //0 norm
           vs(0.0135994,0.005114,6) ) * //0 alluminio
           (0.5);
AnalisiGravita << "B (seconda giornata), senza peso: (b_norm + b_alluminio)/2" << endl;
AnalisiGravita << b_np << endl << endl;

//Media Coefficienti rette con peso
vs b_conp = (vs(0.0282376,0.001601,6) + //0 peso
            vs(0.0395786,0.001124,6) ) * //0 pesoalluminio
            (0.5);
AnalisiGravita << "B (seconda giornata), con peso: (b_peso + b_peso)/2" << endl;
AnalisiGravita << b_conp << endl << endl;

////////////////////////////////////
//Stime gravità
// Velocità media norm (0 gradi)
vs vMed_norm = (vs(0.172473,0.001) + vs(0.170765,0.00122783) ) * 0.5;
vs vMed_allum = (vs(0.121344,0.00126562) + vs(0.117702,0.00208813)) * 0.5;
vs vMed_totnopeso = (vMed_norm + vMed_allum) * 0.5;

// Velocità media col peso (0 gradi)
vs vMed_peso = (vs(0.103018,0.00142386) + vs(0.116131,0.000705806)) * 0.5;
vs vMed_pesoallum = (vs(0.0541008,0.00236734) + vs(0.0751428,0.000329625)) * 0.5;
vs vMed_totpeso = (vMed_peso + vMed_pesoallum) * 0.5;

AnalisiGravita << "Delta G: Rispettivamente a 15, 30, 45 e 45 con peso " << endl;
AnalisiGravita << "15: " << endl << (vMed_totnopeso*b_np) * (1/sin(G15)) << endl;
AnalisiGravita << "30: " << endl << (vMed_totnopeso*b_np) * (1/sin(G30)) << endl;
AnalisiGravita << "45: " << endl << (vMed_totnopeso*b_np) * (1/sin(G45)) << endl;
AnalisiGravita << "45 con peso: " << endl << (vMed_totpeso*b_conp) * (1/sin(G45)) << endl << endl;

vs stimaGravita_corr = (((vs(0.0388864,0.001,7) + vMed_totnopeso*b_np) * (1/sin(G15)) ) + //15 norm
                        ((vs(0.0811784,0.001503,7) + vMed_totnopeso*b_np) * (1/sin(G30)) ) + //30 norm
                        ((vs(0.121616,0.001991,7) + vMed_totnopeso*b_np) * (1/sin(G45)) ) + //45 norm
                        ((vs(0.122322,0.00114,7) + vMed_totpeso*b_conp) * (1/sin(G45)) ) ) * //45 peso
                        (0.25);

```

```
AnalisiGravita << "Stima gravità corretta: G = G0 + DeltaG" << endl;
AnalisiGravita << stimaGravita_corr << endl << endl;

} catch (exception &e) {
cout << e.what() << endl;
return -1;
} catch (string &e) {
cout << e << endl;
return -2;
} catch (const char* e) {
cout << e << endl;
return -3;
}
//cout << "\n";
return 0;
}
```