Studente: Attarantato Kevin
Matricola: 282785

Studente: Giorgia Roselli
Matricola: 282724

Corso di Sistemi Operativi

Progetto per la sessione estiva 2020/2021

“Internet of things”

Docente: Prof. Emanuele Lattanzi

Analisi del problema:

Il problema prevede la realizzazione di un programma multithread che simuli il funzionamento di un sistema Internet Of Things. Il sistema realizzato simulerà una raccolta di informazioni ambientali da un ambiente per mezzo di thread sensori, i quali potranno essere di diverso numero e, quindi, potranno operare anche contemporaneamente.
L'output sarà costituito da alcuni dati statistici, prodotti a seguito del numero prestabilito dei dati di input inseriti dall'utente.
Lo scopo di tale progetto è quello di analizzare e determinare empiricamente gli andamenti della media e della deviazione standard dei tempi di attesa per la lettura dei dati catturati dai sensori da parte degli utenti, al variare del numero di utenti rispetto ai sensori e viceversa.

Specifica dati in input:

Il programma dovrà ricevere in input i seguenti parametri di inizializzazione:

- Acquisizione del numero di utenti;
- Acquisizione del numero di sensori.

Specifica dati in output:

Al termine della simulazione il programma dovrà stampare a video:

- La media e la deviazione standard dei tempi necessari per effettuare la lettura della temperatura e della luminosità da parte degli utenti.

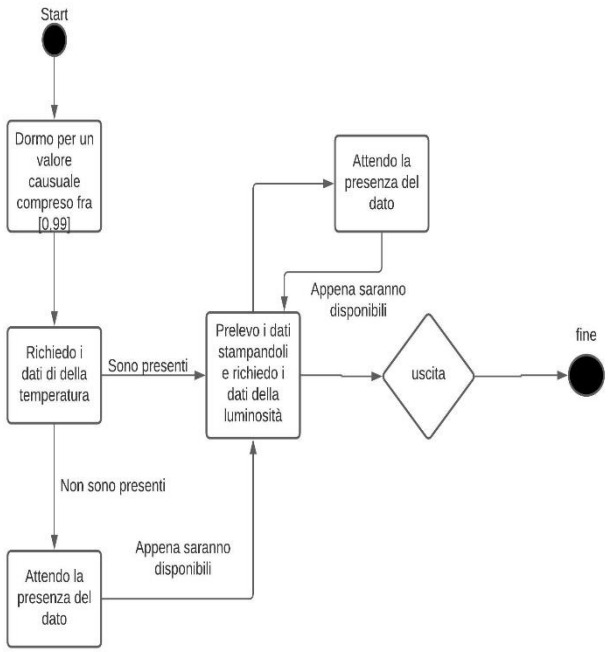
Descrizione della progettazione:

Topologicamente la struttura del programma è composta:

- Dall'oggetto condiviso Cloud, che sarà acceduto dai thread User e Sensor. Inoltre si è pensato di utilizzare il paradigma Produttore-Consumatore per questo oggetto;
- Dall'oggetto condiviso Environment che sarò acceduto dai thread Sensor e WeatherConditioner. Per questo oggetto si è pensato di utilizzare il paradigma Lettori-Scrittori;

L'applicazione inoltre prevedrà un thread Main che avrà la funzione di "regista": creerà le istanze dell'oggetto condiviso e dei thread, avvierà la simulazione, ed una

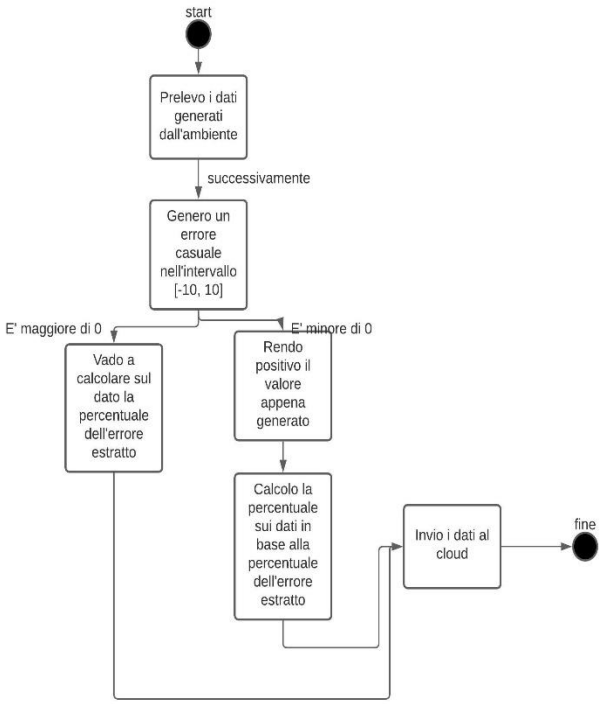
volta che sarà avviato il flusso di esecuzione, dovrà attendere per la terminazione di questi ultimi.



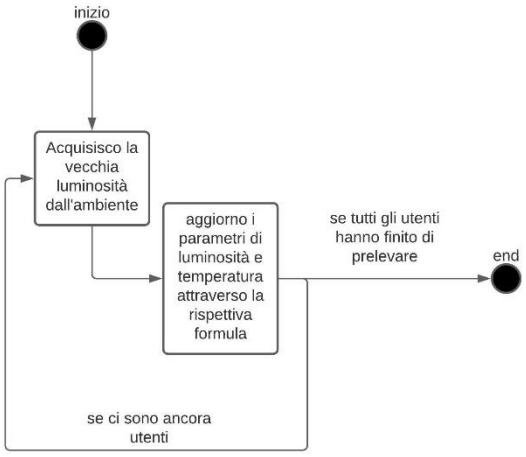
User

Viene emulato attraverso l'uso di un thread.
L'utente dormirà per un tempo casuale compreso fra [0, 99]ms e successivamente andrà a leggere e prelevare i dati dal Cloud. Qual ora i dati non fossero ancora pronti il primo utente che effettuerà la richiesta si fermerà in attesa di quest'ultimi.
In figura a lato è riportato lo schema a blocchi che descrive il comportamento di ogni utente.

Sensor



Viene emulato attraverso l'uso di un thread.
Nella figura qui riportata di lato è possibile visualizzare lo schema a blocchi che descrive il comportamento del thread sensore.
Questo thread prenderà i dati dall'ambiente e successivamente genererà un errore casuale compreso nell'intervallo [-10, 10] il quale nel caso il valore fosse negativo andrà a decrementare i dati presi dall'ambiente pari alla percentuale del valore estratto, in caso di valore positivo, invece, andrà ad incrementare il valore pari alla percentuale del valore estratto.
Successivamente, modificherà i dati in base alla percentuale d'errore inviandoli successivamente al Cloud.

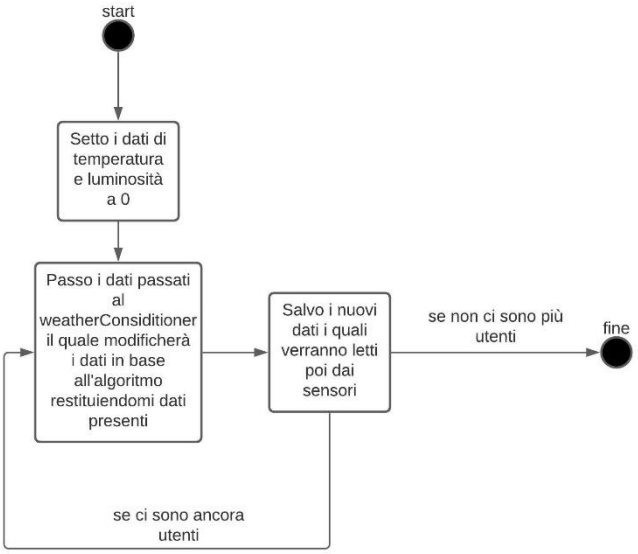


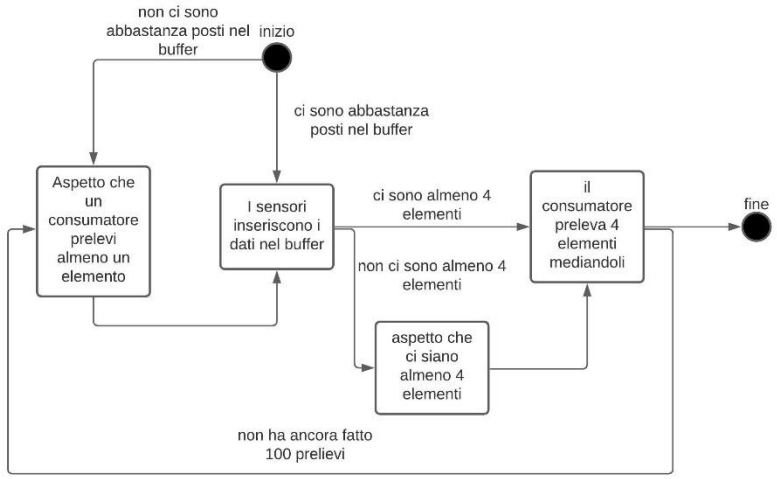
WeatherConditioner

Viene emulato attraverso l'uso di un thread.
Questo thread, semplicemente, per tutta la durata dell'esecuzione, quindi finché ci saranno utenti pronti a richiedere dati prenderà i valori dall'ambiente e gli modificherà in base ad una formula data da specifica.

Environment

L'oggetto Environment è l'oggetto condiviso fra i thread WeatherConditioner e Sensor. Qui a fianco è riportato lo schema a blocchi che rappresenta come nell'oggetto sono emulate le azioni ed i comportamenti dei vari thread mentre modificano i dati, tramite il WeatherConditioner, e vengono misurati dai sensori.





Cloud

L'oggetto Cloud è l'oggetto condiviso fra i thread Utente e Sensore.
Per questo oggetto si è pensato di usare le variabili 'condizione' per la gestione dei permessi così da permettere la sospensione dei thread attivi qual ora non fosse verificata una condizione tra buffer pieno e buffer vuoto.

Descrizione dell'implementazione:

Implementazione della classe Cloud:

La classe **Cloud**, ovvero l'oggetto condiviso fra i thread **Sensor** ed **User**, è stata implementata seguendo le varie analogie con i paradigmi affrontati durante il corso di sistemi operativi.

L'interazione con la classe Sensor è stata così progettata:

- Il metodo pubblico **writeData** dovrà essere in mutua esclusione per la presenza e per la protezione di variabili condivise e bloccante qual ora i buffer risultassero pieni.
  Questo metodo dovrà, quindi, permettere l'inserimento del valore catturato dal sensore in un buffer limitato e circolare avente una grandezza fissa, la quale sarà di 30 posizioni.
  Il metodo disporrà inoltre di due dei buffer appena citati per contenere rispettivamente in uno tutti i valori rappresentati dalla luminosità e nell'altro tutti i valori rappresentati dalla temperatura. Nel caso in cui uno dei due buffer risultasse pieno, come detto prima, fermerà il sensore, invocando il metodo await(), finché non si sarà liberata almeno una posizione dove andare ad inserire il dato. Questo metodo sarà anche necessario per la terminazione deferita dei sensori.

L'interazione con la classe User è stata cosi progettata:

- Il metodo pubblico **readAvarageLight** sarà invocato per andare a leggere e prelevare i dati inerenti alla luminosità.
  Questo metodo dovrà far aspettare, invocando await(), l'utente (consumatore) finché non saranno presenti almeno 4 valori nel buffer circolare. Appena i valori nel buffer saranno disponibili, inizierà a leggerli, eseguirà la media dei 4 valori consecutivi e stamperà quest'ultima;
- Il metodo pubblico **readAvarageTemp** sarà l'ultimo metodo invocato dagli utenti per andare a prelevare i dati inerenti alla temperatura.
  Come nel caso del metodo pubblico precedente questo metodo farà attendere il cliente qual ora non fossero presenti almeno 4 elementi e appena presenti preleverà gli elementi mediandoli e stampandoli.

Si presta all'occorrenza l'utilizzo degli attributi di sincronizzazione quali *semafori binari* e *semafori contatore* utili per garantire l'interazione fra i thread e la mutua esclusione fra quelli concorrenti:

- **lockL** e **lockT** sono i semafori binari a guardia delle variabili condivise **in**, **out** le quali verranno aggiornate di volta in volta essendo i puntatori logici dei buffer;
- **semT** e **semL** sono semafori per la mutua esclusione e per far procedere gli utenti in ordine FIFO;
- **notFullLum**, **notEmptyLum**, **notFullTemp** e **notEmptyTemp** sono semafori condizione (o "condition") che segnaleranno all'utente, quindi al consumatore, che è stato inserito almeno un elemento e al sensore, quindi al produttore, che è stato prelevato almeno un elemento. Oltre a bloccare i due thread in caso di buffer pieno/vuoto.

Questo metodo dispone anche di un ulteriore metodo **getElapsedTime** per la misurazione dei tempi di esecuzione durante quest'ultima.

Implementazione della classe Environment:

La classe **Environment**, ovvero l'oggetto condiviso fra i thread **WeatherConditioner** e **Sensor**, è stata implementata in modo tale da permette la creazione dei valori qui, aggiornarli da WeatherConditioner e fargli misurare dai sensori.

L'interazione con la classe WeatherConditioner è stata cosi progettata:

- Il metodo pubblico **updateParameters** permetterà l'aggiornamento dei parametri tramite le seguenti formule:

  $$Luminosità_{(t1)} = Luminosità_{(t0)} + 1000;\ con\ luminosità\ iniziale = 0$$
  $$Temperatura_{(t1)} = 10 + 0.00022 * Luminosità_{(t1)}$$

  i quali verranno poi restituiti, all'oggetto condiviso Environment, una volta inseriti in un array a due posizioni. Questo metodo sarà anche necessario per la terminazione deferita del thread invocante.

L'interazione con la classe Sensor è stata cosi progettata:

- Il metodo pubblico **misureParameters** dovrà acquisire i valori precedentemente aggiornati dal metodo sopra citato e permetterà, quindi, ai thread sensori di prelevarli per ulteriori modifiche (applicazione di un errore di lettura) prima di essere inseriti nuovamente nel buffer circolare.

Si presta all'occorrenza l'utilizzo degli attributi di sincronizzazione quali *semafori binari* e *semafori contatore* utili per garantire l'interazione fra i thread:

- **Mutex** semaforo binario a guardia della variabile **count**;
- **Wrt** semaforo contatore per la gestione degli "scrittori", ovvero il WeatherConditioner.

Questo metodo dispone inoltre di altri due metodo **getLum()** e **getTemp()** per prendere i valori ottenuti dalla modifica della classe WeatherConditioner.

Implementazione della classe User:

Il thread che emula l'utente, continuerà a prelevare dati fino a ripetere questa azione 100 volte. Dopo ogni lettura, inoltre, dormirà per un numero casuale compreso nell'intervallo [0,99].

Implementazione della classe Sensor:

Il thread che emula i sensori verrà istanziato nella main è continuerà a misurare i parametri dall'ambiente finché non sarà terminata la simulazione. L'interruzione del thread sarà deferita, quindi attraverso un *interrupt*.

Implementazione della classe WeatherConditioner:

Questo thread verrà istanziato nella main e per tutta la durata della simulazione aggiornerà i dati secondo la formula riportata nella descrizione della classe Environment. L'interruzione del thread sarà, perciò, deferita, ovvero con l'utilizzo dell'*interrupt*.

Implementazione della classe Main:

L'implementazione della struttura della Classe Main comprende un'introduzione all'inserimento dei dati di input, con la stampa a video dei relativi messaggi inerenti al numero richiesto di Utenti e Sensori.
L'acquisizione degli input da tastiera è gestita dalla classe Scanner.
Successivamente sono state create le istanze dei thread e dell'oggetto condiviso, per poi essere inizializzati. Le strutture **try/catch** sono state utilizzate per la gestione di eventuali eccezioni. All'avvio della simulazione è possibile osservare l'interazione tra i singoli thread.
Nella suddetta classe, inoltre, sono state definite le funzioni per il calcolo della **media** e della **deviazione standard** dei tempi necessari per effettuare la lettura dei dati da parte degli utenti mediante i seguenti passi:

1) Calcolo la media andando a sommare tutti i tempi medi effettuati dagli utenti e dividendo il risultato per il numero di utenti

```
// calcolo del tempo medio
for(int i = 0; i < utenti.length; i++){
    tempoMedioTotale += utenti[i].tempoMedioPerUtente;
}
// il risultato appena ottenuto lo divido per gli utenti e lo salvo nella corrispondente variabile
double tempoMedio = tempoMedioTotale/utenti.length;
```

2) Calcolo successivamente la varianza andando a ricavare per prima cosa il numeratore della varianza come segue:

numeratore varianza = ∑ (*tempo medio di un utente – tempo medio degli utenti*)²

e, una volta ottenuto il numeratore, vado a calcolare la deviazione standard in base alla seguente formula:

$$\text{deviazione standard} = \sqrt{\frac{numeratore\ varianza}{numero\ degli\ utenti}}$$

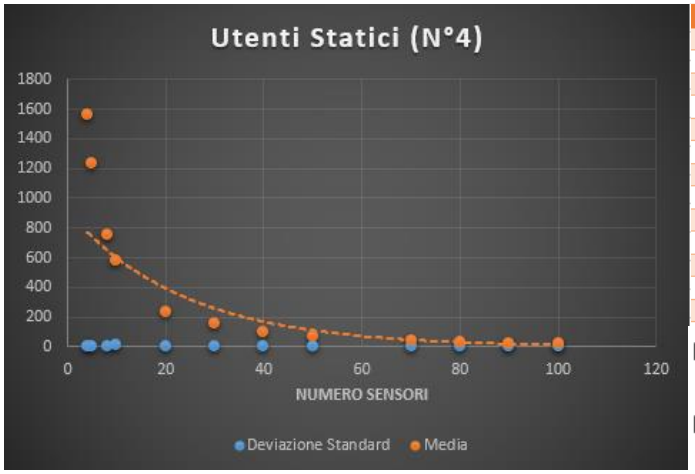
```
// vado a calcolare il numeratore della varianza attraverso la libreria matematica
for(int i = 0; i < utenti.length; i++){
    numeratoreVarianza += Math.pow((utenti[i].tempoMedioPerUtente - tempoMedio),2);
}
// calcolo ora la deviazione standard sempre tramite la libreria matematica
deviazioneStandard = Math.sqrt(numeratoreVarianza/utenti.length);
```

Contributo degli studenti:

Abbiamo, per quanto riguarda la creazione del codice e la stesura della relazione, sempre voluto cooperare affinché il lavoro venisse svolto nel miglior modo possibile perciò non c'è stata una vera e propria divisione del carico di lavoro o delle varie componenti; Tuttavia è possibile dire che lo studente Attarantato si sia impegnato nella stesura delle classi Environment, WeatherConditioner, Sensor e parte della classe Cloud e nel reperire i dati necessari all'ultima sezione della relazione oltre che alla scrittura dei primi due paragrafi della relazione, mentre la studentessa Roselli si è impegnata più nella stesura delle classi User e Cloud e nel calcolo della varianza oltre che nella scrittura dell'ultimo paragrafo della relazione.

Fase di test del programma:

Test n. 1 – Andamento del tempo medio impiegato dagli utenti per ottenere la lettura dei dati al variare del numero di sensori



UTENTI STATICI (N°4)		
SENSORI	DEVIAZIONE STANDARD	MEDIA
4	4,21	1563
5	8,08	1234
8	5,25	755,88
10	9,06	582,2
20	6,01	236,67
30	3,51	156,49
40	3,41	105
50	1,75	74,84
70	0,8	40,34
80	0,75	32,35
90	0,51	25,73
100	0,61	22,16

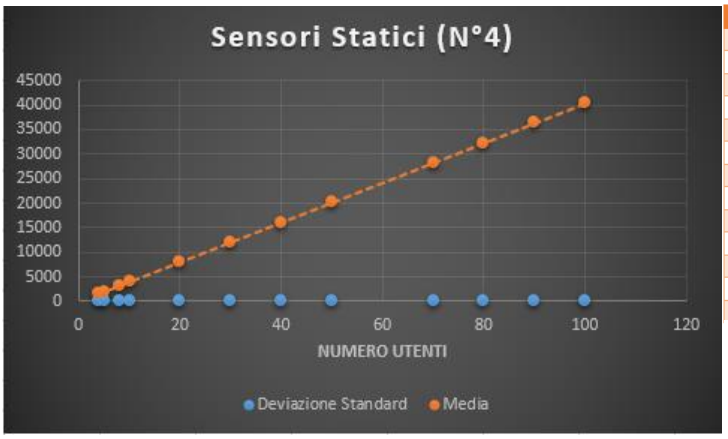
Parametri fissati:

Utenti = 4

Parametri variabili:

Sensori = [4, 100].

Test n.2 – Andamento del tempo medio impiegato dagli utenti per ottenere la lettura dei dati al variare del numero di utenti



SENSORI STATICI (N°4)		
UTENTI	DEVIAZIONE STANDARD	MEDIA
4	7,46	1563,55
5	7,21	1978,23
8	11,58	3180,29
10	13,76	3990
20	27,87	8033,49
30	38,48	12084
40	53,89	16127
50	62,88	20175,66
70		
80	95,88	32299
90	112,24	36515,94
100	117,24	40390,15

Parametri fissati:

Sensori = 4

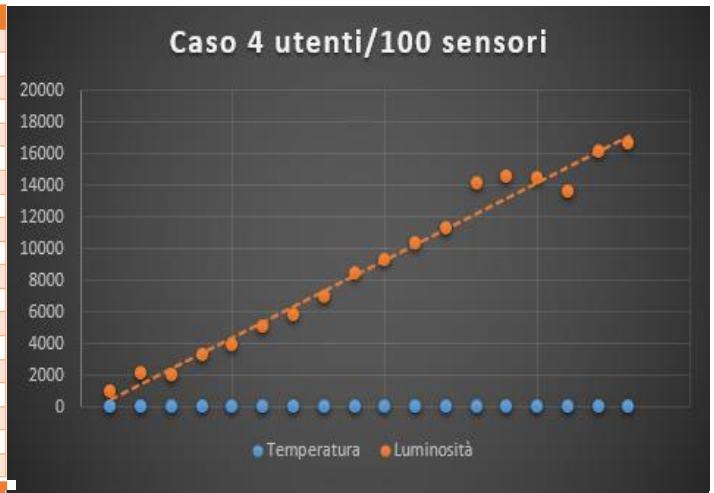
Parametri variabili:

Utenti = [4, 100].

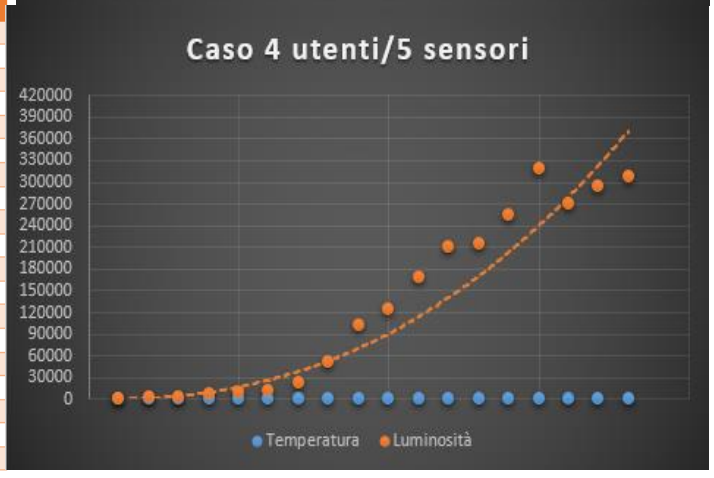
Test n.3 – Coppia di grafici che mostrino la relazione che sussiste tra luminosità e temperatura misurate da un sensore nel caso di:

- **4 utenti e 100 sensori;**
- **4 utenti e 5 sensori.**

4 UTENTI / 100 SENSORI			
UTENTI	SENSORI	LUMINOSITA'	TEMPERATURA
100	4	1020	10.2
		2100	10,73
		2000	10,22
		3300	11,48
		3880	10,34
		5050	10,99
		5820	10,77
		6930	11,21
		8480	12,23
		9300	11,14
		10340	11,47
		11280	11,67
		14170	13,78
		14560	13,37
		14400	13,43
		13650	11,9
		16150	12,56
		16660	13,25



4 UTENTI / 5 SENSORI			
UTENTI	SENSORI	LUMINOSITA'	TEMPERATURA
4	5	900	9
		2040	10,42
		2910	10,13
		6930	11,21
		10560	11,71
		12480	12,92
		22000	14,62
		51840	21,97
		101520	32,9
		125460	37,58
		168950	47,83
		211050	56,7
		215760	56,56
		254800	65,64
		317900	80,7
		269360	68,16
		293910	74,15
		307020	77,52



Per quanto riguarda il primo test ed il secondo, l'indice di dispersione consente di misurare l'estensione di una distribuzione. Per quantificare questa dimensione è stato utilizzato lo ***scarto quadratico medio*** (oppure ***deviazione standard***), prodotto in output dal programma.
Esso è dato dalla radice quadrata della media dei quadrati degli scarti, in altre parole dalla distanza media di tutti i punti dalla media. Prima si trova la media, poi si trovano gli scarti, cioè quanto distano queste osservazioni dalla media. Vengono elevati tutti i risultati al quadrato per forzare il segno positivo ed effettuiamo la media. Il risultato ottenuto è la ***varianza***, ma quest'indice è poco trasparente perché si ottiene un numero troppo elevato da confrontare con il data set. Infatti al risultato viene applicata la radice quadrata, trovando così la ***deviazione standard***: essa ci dice quanto si discostano i valori dalla media, in media, misurando la dispersione di una distribuzione. Infine la deviazione standard permette di dividere la gaussiana in intervalli tipici.