RELAZIONE sul PROGETTO
per l'ESAME di
TECNICHE di ANALISI NUMERICA e SIMULAZIONE

Simulazione di un rivelatore di vertice

Anno Accademico 2018/2019

Professore: MASERA MASSIMO

Candidate: RACCA ELEONORA
SAUDA CRISTINA

1 Introduzione

Il progetto consiste nello sviluppo di una simulazione di un rivelatore di vertice di un esperimento a collider. Il rivelatore è costituito da una struttura cilindrica concentrica formata da una beam pipe di berillio e due layer di rivelatori al silicio.
Il programma consta di tre parti: una fase di generazione degli eventi, una di ricostruzione e infine una di analisi. Il codice è stato scritto in linguaggio C++ usando i pacchetti del software ROOT e le classi già presenti per implementare le classi scritte per il progetto.

2 Struttura del programma

Il programma si articola su tre macro principali, due macro per l'esecuzione e alcune classi, che definiscono gli oggetti usati nello sviluppo del progetto. Le classi utilizzate sono le seguenti:

- `Rivelatore`, che contiene le informazioni geometriche e fisiche dell'apparato.
- `Punto`, che contiene le coordinate nei tre sistemi di riferimento, cartesiano, cilindrico e sferico.
- `Urto`, che eredita da `Punto` e implementa i metodi per generare urti sulla beam pipe e sui layer, effettuare lo smearing e generare i punti di rumore.
- `Vertice`, che eredita da `Punto`, memorizza le coordinate del vertice e la molteplicità dell'evento. Implementa un metodo per ricostruire la coordinata del vertice a partire da due urti presenti sui layer di rivelazione.
- `Trasporto`, che contiene la direzione di propagazione dei singoli urti e permette la sua rotazione angolare in caso di scattering multiplo.

Per compilare ed eseguire il programma con le configurazioni di default è necessario seguire questa procedura:

1. scaricare la repository GitHub[1]
2. da terminale posizionarsi dentro la directory della repository e lanciare i seguenti comandi:

```
cd Provette
root
```

3. all'interno di ROOT eseguire i seguenti comandi

```
.x compila.C
EsecuzioneEsame()
```

La funzione `EsecuzioneEsame` è contenuta nella macro *EsecuzioneEsame.C*, il cui compito è quello di richiamare le macro di generazione *Albero.C*, ricostruzione *Ricostruzione.C* e analisi *Analisi.C*.
La funzione `EsecuzioneEsame` ha come argomenti tre parametri booleani, che permettono di specificare se eseguire simulazioni multiple su differenti livelli di rumore e se effettuare generazione e ricostruzione a partire file di configurazione oppure di impostare i parametri a mano.
In ognuno dei casi, la singola simulazione è formata da tre stadi: generazione, ricostruzione e analisi. Di default viene eseguita andando a leggere le informazioni necessarie in due file di tipo testo. Il file *Configurazioni/Generazione.txt* contiene le informazioni riguardanti il numero di eventi da generare, la distribuzione della molteplicità degli eventi, la presenza o meno dello scattering multiplo e la distribuzione di pseudorapidità η. Il file *Configuraioni/Ricostruzione.txt* contiene le informazioni necessarie alla macro di ricostruzione per abilitare il rumore e specificare quale distribuzione segue la sua molteplicità.
Nel caso di esecuzioni multiple, la fase di generazione è unica, mentre ricostruzioni e analisi sono differenti per ogni livello di rumore, definiti nei file *ConfigurazioniRicostruzioneRumoreN.txt*. Una volta

[1] https://github.com/eleoracca/Alacre

eseguite tutte le simulazioni richieste, viene invocata la funzione `PostAnalisi`, che permette di comparare tramite dei `MultiGraph` i grafici di efficienza e risoluzione fatti nelle analisi precedenti e riportati nella Sezione 3.

La macro *Albero.C* è quella che si occupa della generazione degli eventi. Viene creato il `Tree` "gaggia" con quattro `Branch`: Vertice, UrtiBeamPipe, UrtiRivelatore1 e UrtiRivelatore2. Il primo contiene oggetti di classe `Vertice`, mentre gli altri tre sono dei `TClonesArray` di classe `Urto`.
All'interno della macro viene letto il file di generazione, che contiene i parametri per la simulazione. Per la molteplicità delle particelle nell'evento sono supportate le distribuzioni gaussiana, uniforme, fissa oppure quella contenuta nel file *Configurazioni/kinem.root*. Per ogni evento, viene scelto il numero di particelle mediante la funzione `DecisioneMolteplicita` e viene generato il vertice tramite l'apposito costruttore.
Il processo di generazione consiste principalmente in due cicli `for` annidati: il primo gira sul numero di eventi, il secondo sul numero di particelle per evento.
Per ogni particella da generare, viene calcolata la direzione di propagazione per l'angolo ϕ con una distribuzione uniforme tra $[0, 2\pi]$ e in θ con la funzione `EtaTheta`. Nel file testuale è specificato se utilizzare una distribuzione di pseudorapidità uniforme o la distribuzione contenuta in *Configurazioni/kinem.root*, da cui si ricava il valore di θ.
Infine, vengono generati l'urto sulla beam pipe tramite il metodo `UrtoDaVertice` e gli urti sui due layer tramite il metodo `UrtoDaUrto`. In caso sia attivo lo scattering multiplo, la funzione `UrtoDaUrto` aggiorna la direzione di propagazione in base alla formula e ai parametri del rivelatore.
Le posizioni del vertice e dei punti vengono salvate nel `Tree` ed esportate nel file *Output/Simulazione.root*.

La macro *Ricostruzione.C* legge dal `Tree` "gaggia" i `Branch` UrtiRivelatore1 e UrtiRivelatore2. Successivamente crea il `Tree` della ricostruzione "rovere", nel quale vengono salvati gli urti ricostruiti dopo l'applicazione dello smearing gaussiano ed eventualmente dello scattering multiplo in due `Branch` per i layer del rivelatore e in un terzo `Branch` salva il numero di punti di rumore generati per ogni evento.
Si cicla su tutti gli eventi della simulazione e per ognuno viene applicato lo smearing gaussiano con la funzione `Smearing`, che, iterando sugli urti per ogni layer, applica lo smearing tramite la funzione `SmearingGaussiano`, metodo di `Urto`. In caso gli urti della generazione siano fuori dall'accettanza del rivelatore, i punti non vengono salvati nell'array dell'evento.
In seguito, viene aggiunto il rumore tramite due possibili funzioni a seconda della distribuzione: `RumoreGaussiano`, che genera un numero di punti con distribuzione gaussiana, e `RumoreFissa`, che genera un numero fissato di punti. I punti di rumore vengono aggiunti in fondo ai TClonesArray dei due layer.
Infine il `Tree` viene salvato nel file *Output/Ricostruzione.root* nel caso di singola simulazione oppure nei file *Output/RicostruzioneRumoreN.root* per simulazioni multiple.

La macro *Analisi.C* si occupa di stimare la coordinata z del vertice a partire dagli urti ricostruiti e di confrontarla con il suo valore generato leggendo i `Tree` "gaggia" e "rovere" e relativi `Branch`.
Per ogni evento si valutano tutte le possibili coppie che si possono ottenere scegliendo un punto ricostruito sul layer 1 e uno sul layer 2. Per ogni coppia di punti si valuta la differenza $\Delta\phi$: se questa è minore di una soglia impostata dall'utente, allora viene invocato il metodo `TrovaVertice` di `Vertice` per calcolare tutti i possibili vertici ricostruiti dell'evento con le tracce ipotizzate.
A questo punto, si valuta quale sia il vertice più probabile tramite la funzione `Moda`. Essa cerca il primo bin dell'istogramma in cui è presente il valore massimo di conteggi e assegna alla $z_{ricostruita}$ il valore centrale del bin (esempi in Figura 2 e Figura 3). In seguito, controlla che non siano presenti altri massimi di stessa altezza a destra del picco trovato: in caso negativo, viene restituito il valore precedentemente trovato. In caso contrario, viene valutata la distanza tra i due bin: se piccola, viene fatta la media tra i due, altrimenti viene scelto il valore più vicino alla coordinata $z = 0$. In Figura 4 è riportato un esempio di tale situazione.
Infine, si valutano efficienza e risoluzione della simulazione tramite la costruzione di grafici. Tut-

to viene salvato nel file *Output/Analisi.root* in caso di simulazione singola oppure *Output/Analisi_RumoreN.root* per simulazioni multiple.
In caso di simulazioni multiple, inoltre, viene invocata la funzione `PostAnalisi`, che combina i grafici di efficienza e risoluzione per effettuare un confronto al crescere del rumore. Per stampare i grafici, prodotti e salvati nei relativi file *.root* durante l'esecuzione del programma, è stata implementata la macro *generaGrafici.C*.

3 Risultati

Di seguito vengono riportati i grafici prodotti durante l'analisi.
In Figura 1 viene presentata la distribuzione degli scarti $(z_{ricostruita} - z_{vero})$ lungo l'asse z.

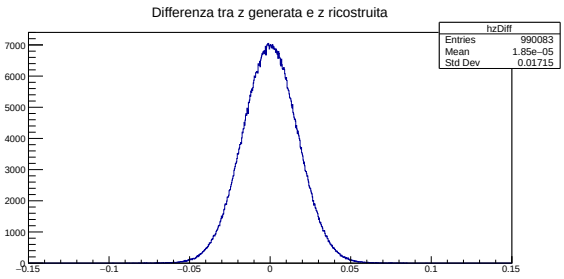

Figura 1: *Distribuzione degli scarti sulla variabile z*

Come spiegato precedentemente, per valutare la coordinata z del vertice ricostruito per ogni evento, è stato costruito un istogramma con tutti i possibili vertici dell'evento. Viene poi valutata la moda della distribuzione e presa come coordinata del vertice. La Figura 2 e Figura 3 sono due esempi di distribuzione in cui la moda è ben definita. Nel primo caso tutti i possibili vertici sono raggruppati attorno al valore medio; nel secondo le possibili coppie di punti ricostruiscono il vertice sia attorno al picco, sia in posizioni più distanti.

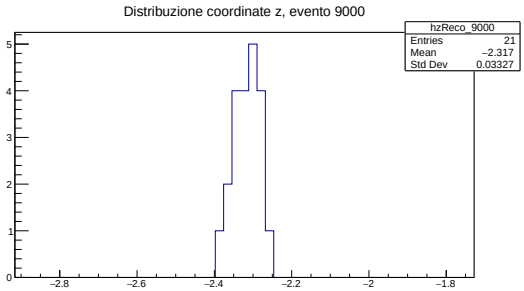

Figura 2: *Istogramma per valutare la moda della distribuzione dei vertici ricostruiti*

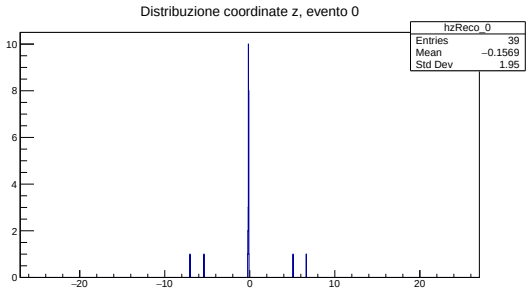

Figura 3: *Istogramma per valutare la moda della distribuzione dei vertici ricostruiti*

Invece, in Figura 4 si può vedere che la distribuzione è bimodale. In questo caso la funzione `Moda` effettua una media dei due massimi se essi sono sufficientemente vicini, altrimenti prende il valore del massimo più vicino a $z = 0$.

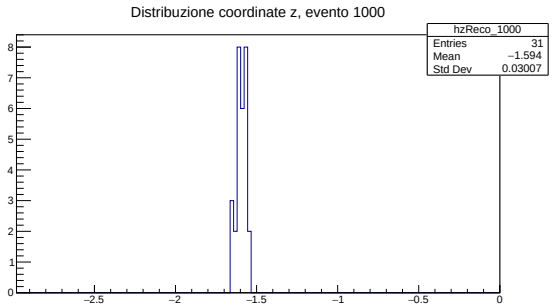

Figura 4: *Istogramma per valutare la moda della distribuzione dei vertici ricostruiti*

Alcune ricostruzioni però possono non dare risultati. In Figura 5 viene riportato il caso in cui non è presente una moda della distribuzione, perché tutte le possibili coppie portano a ricostruire un vertice diverso. In questo caso, il vertice non viene ricostruito.
In Figura 6 viene riportato un grafico vuoto: questo è dovuto al fatto che nessuna delle coppie possibili riesce a ricostruire un possibile vertice.

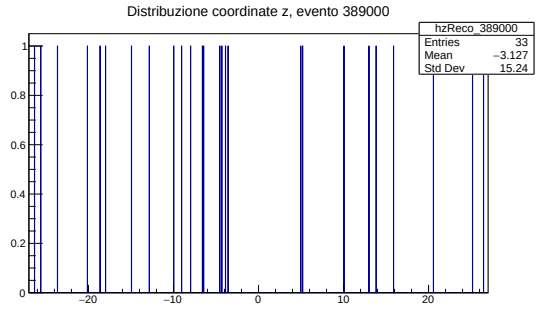

Figura 5: *Istogramma per valutare la moda della distribuzione dei vertici ricostruiti*

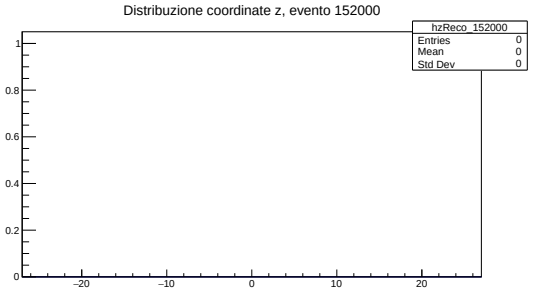

Figura 6: *Istogramma per valutare la moda della distribuzione dei vertici ricostruiti*

Infine vengono proposti alcuni confronti di ricostruzione in presenza di diverse magnitudini di rumore. Sono state effettuate quattro ricostruzioni e analisi con livelli di rumore crescente: 0, 10, 50 o 100 punti di rumore per ogni layer. In Figura 7 viene riportato il confronto della risoluzione dell'apparato in funzione della posizione del vertice dell'evento, mentre in Figura 8 si può osservare come varia la risoluzione dell'apparato in funzione della molteplicità, entrambi all'aumentare del rumore sui due layer.

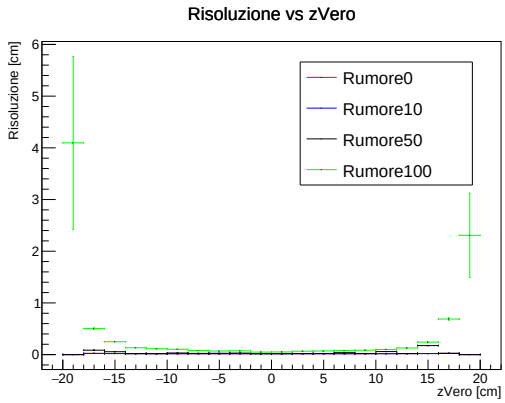

Figura 7: *Confronto della risoluzione dell'apparato in funzione della posizione del vertice*

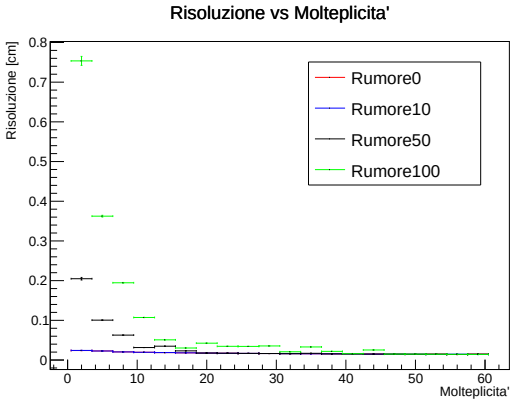

Figura 8: *Confronto della risoluzione dell'apparato in funzione della molteplicità*

In Figura 9 e Figura 10 vengono riportati anche i grafici di confronto in assenza dell'analisi con 100 punti di rumore per layer, per poter osservare meglio le curve che altrimenti vengono schiacciate verso l'asse delle ascisse a causa della curva a rumore maggiore.

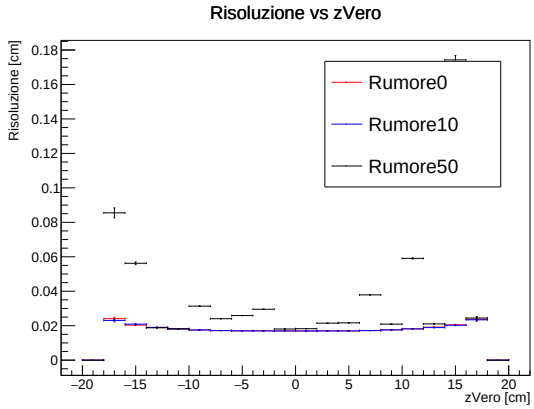

Figura 9: *Confronto della risoluzione dell'apparato in funzione della posizione del vertice escludendo la curva per 100 punti di rumore per layer*

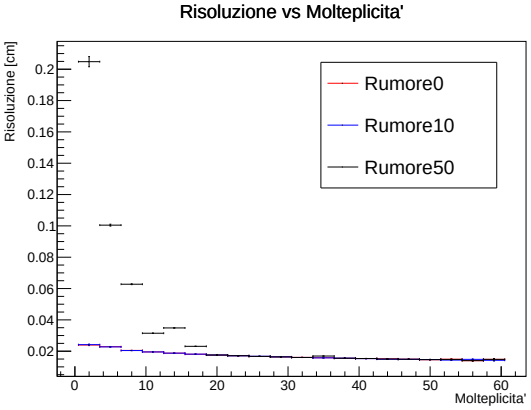

Figura 10: *Confronto della risoluzione dell'apparato in funzione della molteplicità escludendo la curva per 100 punti di rumore per layer*

Analogamente, vengono riportati anche i confronti al variare del rumore per l'efficienza di ricostruzione dell'apparato in funzione della molteplicità dell'evento. In Figura 11 viene valutata l'efficienza di ricostruzione su tutti gli eventi della simulazione, mentre in Figura 12 il confronto escludendo la ricostruzione con 100 punti di rumore. In Figura 13 e Figura 14 si valuta l'efficienza selezionando solo gli eventi che hanno una generazione del vertice entro 1σ da $z = 0$ e in Figura 15 e Figura 16 l'efficienza selezionando gli eventi entro 3σ da $z = 0$.

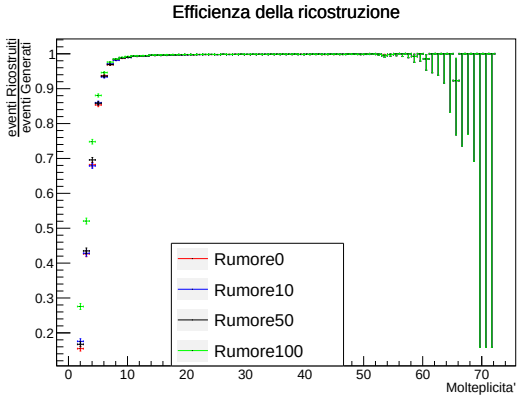

Figura 11: *Efficienza di ricostruzione in funzione della molteplicità valutando tutti gli eventi*

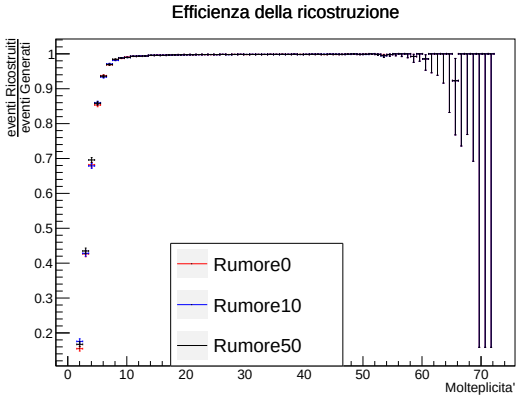

Figura 12: *Efficienza di ricostruzione in funzione della molteplicità valutando tutti gli eventi escludendo l'analisi con 100 punti di rumore*

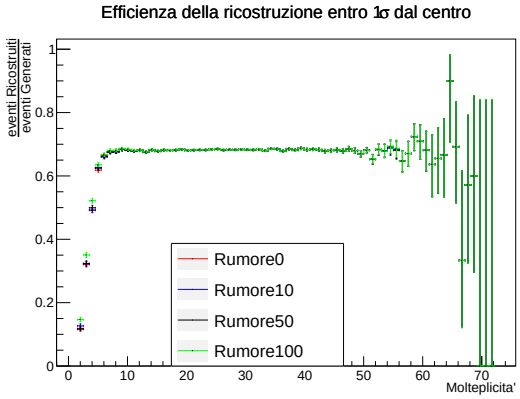

Figura 13: *Efficienza di ricostruzione in funzione della molteplicità valutando gli eventi entro 1σ*

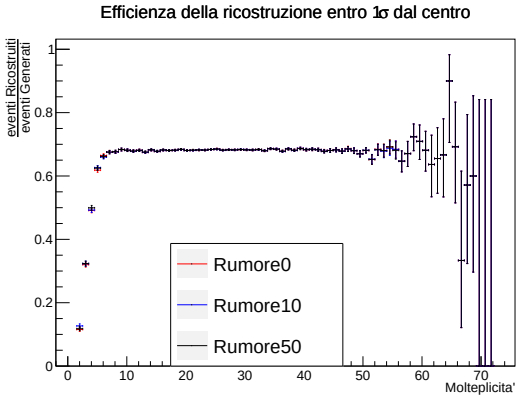

Figura 14: *Efficienza di ricostruzione in funzione della molteplicità valutando gli eventi entro 1σ escludendo l'analisi con 100 punti di rumore*

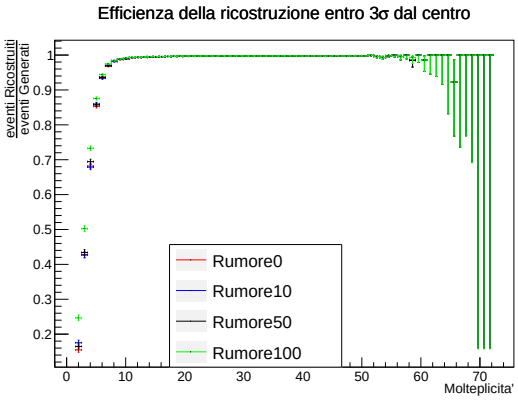

Figura 15: *Efficienza di ricostruzione in funzione della molteplicità valutando gli eventi entro 3σ*

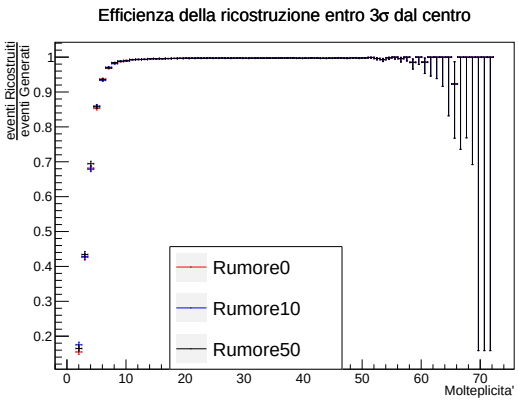

Figura 16: *Efficienza di ricostruzione in funzione della molteplicità valutando gli eventi entro 3σ escludendo l'analisi con 100 punti di rumore*