Simulazione e ricostruzione di vertice

Maria Susini, Elisa Visentin

Scopo del programma è studiare la propagazione delle particelle cariche generate dalla collisione di due fasci di particelle attraverso un apparato sperimentale costituito dalla beam pipe e da due layer di rivelatori. Segue la descrizione della routine di esecuzione del codice e della sua struttura. In conclusione sono riportati i risultati ottenuti.

---

1 Esecuzione del codice

Per avviare il programma completo di simulazione e ricostruzione occorre seguire le seguenti istruzioni:

- .L compilevertex.C
- compilevertex(): comando per compilare le classi e le macro custom necessarie ad eseguire la simulazione, la ricostruzione e la successiva analisi
- simulazione(): comando per eseguire la parte di simulazione e salvare i risultati nel tree in "simulazione.root"
- ricostruzione(): comando per eseguire la ricostruzione di vertice per ogni evento generato e salvare i risultati del processamento nel file "ricostruzione.root" (contiene anche la funzione *plot* utilizzata per analizzare i risultati)
- canvasses(): comando per disegnare gli istogrammi che rappresentano i risultati ottenuti.

2 Descrizione classi e macro implementate

In questo paragrafo si descrivono le caratteristiche principali delle classi e delle macro utilizzate per la simulazione e la ricostruzione. Le classi implementate sono *Vertex*, *Track* e *Intpoint*; le macro sono *simulazione*, *ricostruzione* e *canvasses*.

2.1 Classe Vertex

La classe *Vertex* viene utilizzata per generare le tre coordinate cartesiane del vertice di interazione, la molteplicità (numero di particelle cariche prodotte a seguito della collisione) e le relative direzioni di propagazione.
I data member di questa classe sono le coordinate cartesiane *fX,fY,fZ* del vertice, la molteplicità ed esso associata *fMulti* e gli angoli polare e azimutale *fTheta* e *fPhi* che indicano la direzione di propagazione della particella carica generata. Fanno parte dei member function private la funzione *Var*, che permette di impostare i valori delle coordinate cartesiante tramite Box-Muller, e le funzioni *Multunif* e *Multfunc* con le quali è possibile fissare la molteplicità associata al vertice. Nel primo caso la molteplicità è estratta da una distribuzione uniforme, nel secondo da una distribuzione assegnata rappresentata da un istogramma.
Per quanto riguarda le member function pubbliche, oltre ai costruttori di default, standard e copia, si hanno delle funzioni *Get* che sono utilizzate per accedere ai data members. Si ha poi la funzione *Initialdir* per inizializzare il valore di *fTheta* e *fPhi*; il primo è assegnato a partire da una distribuzione nota in pseudorapidità e il secondo è estratto uniforme nell'intervallo $[0,2\pi]$.

2.2 Classe Track

La classe *Track* viene utilizzata per generare la traccia della particella carica noti gli angoli di propagazione iniziali e individuare il suo punto di intersezione con i diversi layer di cui è composto l'apparato sperimentale.
I data member di questa classe sono i coseni direttori *fC1*, *fC2* e *fC3* della retta e gli angoli *fTheta* e *fPhi* che le funzioni implementate nella classe aggiornano.
Per quanto riguarda le member function, si ha una funzione *Parameter* per stimare il valore del parametro t della retta. Questa viene poi utilizzata nella funzione *Intersection* al fine di valutare il valore delle coordinate del punto di intersezione fra la retta e la beam pipe o il piano di rivelazione. Con la funzione *Intersection2* si valuta invece la coordinata z dell'intersezione fra una tracklet e il piano $x = 0$.
In questa classe è anche implementato il metodo necessario per considerare l'influenza del multiple scattering sulla propagazione delle particelle cariche: tramite la funzione *Multiplescattering*, che può essere abilitata o meno tramite una variabile booleana, si assegnano nuovi valori per *fTheta* e *fPhi*. Questi andranno a rappresentare la direzione di propagazione delle particelle cariche a seguito del multiple scattering.
Sono presenti anche due funzioni *Get* per accedere al valore degli angoli.

2.3 Classe Intpoint

La classe *Intpoint* viene utilizzata per salvare le coordinate del punto di intersezione (gli elementi di questa classe sono gli oggetti poi salvati nei tree).
I data member sono le coordinate del punto di intersezione *fX*, *fY* e *fZ*. Si ha inoltre una variabile *fLabel* per salvare il numero di traccia a cui appartiene l'intersezione.
Come member function si hanno le funzioni *Get*, utlizzate per accedere al valore dei data member. La funzione *Smearing* viene impiegata in fase di ricostruzione per applicare lo smearing ai punti di intersezione simulati (lo smearing serve per considerare la granularità dell'apparato).
E' presente anche un costruttore *Intpoint* per generare i punti di noise distribuiti uniformemente sulla superficie dei rivelatori.

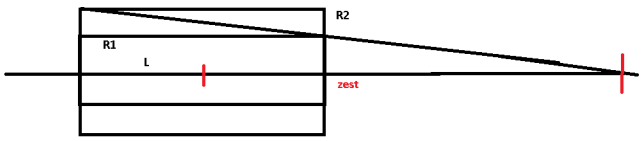
2.4 Macro simulazione

La macro *simulazione.C* rappresenta la parte del programma finalizzata all'ottenimento dei dati simulati; nel caso preso in esame per ogni evento si vogliono ottenere i punti di intersezione di ciascuna particella generata con i layer di rivelatori.
Nella parte iniziale della macro sono definite alcune variabili come *const*: sono specificate le dimensioni fisiche dell'apparato sperimentale (raggio dei vari layer e lunghezza), il valore dei parametri di disperione da utilizzare nella generazione dei punti di interazione primari e il numero di eventi di collisione *kEvents*. Si ha anche la stringa *kMul* per scegliere quale distribuzione di molteplicità si vuole utilizzare (il valore "sì" indica l'utilizzo della distribuzione assegnata, "no" di quella uniforme). E' stata inserita anche la variabile boolenana *kMs* per attivare e disattivare il multiple scattering (true=ms attivo). Si possono inoltre fornire i valori limite in pseudorapidità e il periodo che indica ogni quanti eventi stampare a schermo. I parametri della simulazione sono passati alla macro di ricostruzione tramite un file *.txt*.
Il programma di simulazione genera quindi *kEvents* eventi che vegono salvati in un tree costituito da quattro branches: nel primo sono salvate le coordinate e la molteplicità relative al vertice primario e nei successivi tre sono contenuti TClonesArray di oggetti *Intpoint* utilizzati per salvare le coordinate dei punti di intersezione della particella con la beampipe e i due piani traccianti considerando il multiple scattering. Si salvano tutti gli hit, compresi quelli di tracce che colpiscono un solo piano di rivelazione, e tutti i vertici primari generati.
Il tree così ottenuto viene salvato nel file *simulazione.root*.

2.5 Macro ricostruzione

La macro *ricostruzione.C* contine due funzioni:

- *ricostruzione()*: è utilizzata per applicare smearing e noise ai dati simulati e ricostruire la coordinata z del vertice primario. Si inizia dalla lettura dei dati del tree contenuto in *simulazione.root* e dall'applicazione dello smearing ai punti di intersezione; vengono scartati quelli che risultano esterni alla dimensione del rivelatore. Si prosegue applicando i punti di noise e ricostruendo la coordinata z del vertice primario tramite tracklet: per ogni evento si vanno a valutare tutte le combinazioni possibili fra punti che si trovano sul primo e secondo layer applicando la condizione che la loro differenza angolare sia minore di *phimax*, ovvero la massima differenza angolare fra le due hit di una traccia (valutata per eccesso come pari a 10 milliradianti dall'osservazione dei risultati ottenuti per tale differenza angolare da una simulazione indipendente). Si prosegue valutando l'esistenza di un punto di intersezione fra tracklet e piano $x = 0$; in caso positivo si riempiono un istogramma e il vector *Zintersection* con la coordinata z dell'intersezione ottenuta.
  In figura è rappresentata la sezione longitudinale del tracciatore; dal suo studio si ricava il valore estremo in z impiegato negli istogrammi utilizzati per valutare la coordinata z ricostruita: il valore massimo per z dato dall'intersezione di una tracklet e l'asse dei fasci si ottiene dalla congiunzione dei due estremi opposti nei tracciatori. Si ottiene quindi la relazione:
  $$zest = R2 * \left(\frac{L}{(R2 - R1)} \right) - \frac{L}{2}$$

  Si vuole inoltre specificare che l'algoritmo è stato costruito in modo da selezionare il bin relativo al primo massimo dell'istogramma per determinare il valore di *zrec* in caso sia presente più di un massimo.

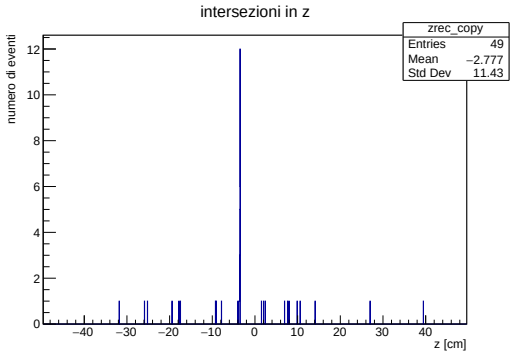
- *plot()*: è impiegata per lo studio dell' efficienza e risoluzione dell'algoritmo di ricostruzione. Le vengono passati dei vector contenenti la coordinata z dei punti ricostruiti e quella del vertice primario z_{true}, la molteplicità e l'errore in z del vertice ricostruito. Con queste informazioni si riempono gli istogrammi relativi ai residui, all'efficienza rispetto alla molteplicità e a z_{true} e alla risoluzione rispetto alla molteplicità e z_{true}. I risultati ottenuti sono salvati nel file *istogrammi.root*.

2.6 Macro canvasses

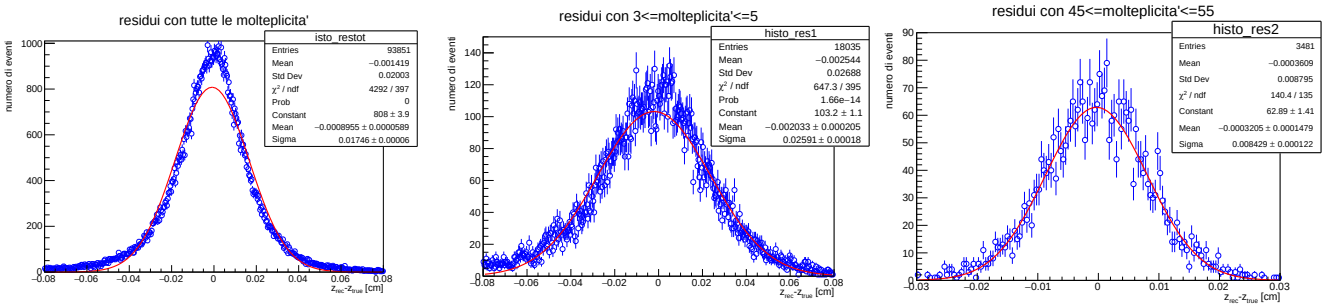
La macro *canvasses.C* è utilizzata per eseguire i plot degli istogrammi contenuti in *istogrammi.root* e applicare un fit gaussiano sui residui.
Questa macro contiene anche la funzione *SetmyIsto()* per impostare stile, titoli e range degli istogrammi.

3 Risultati

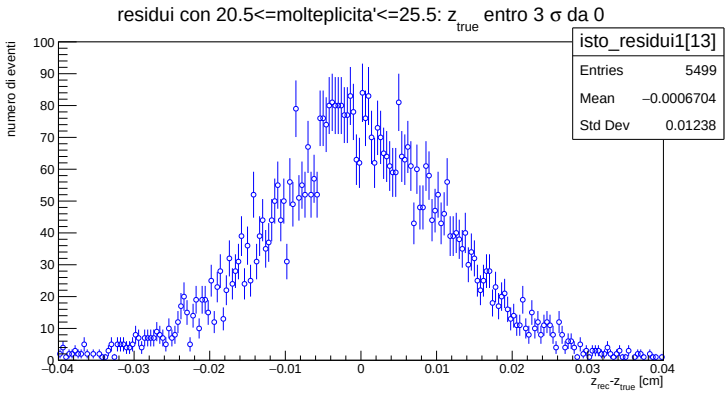
A titolo esemplificativo si riporta di seguito uno degli istogrammi utilizzati per valutare il valore di z_{rec}. Tale istogramma è riferito a una simulazione di 100000 eventi in cui è stata considerata come distribuzione di molteplicità quella assegnata.



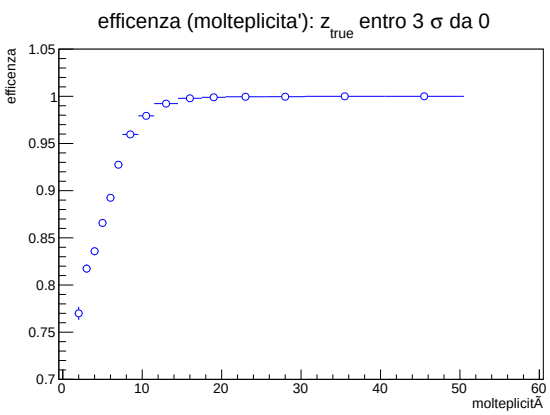
Per quanto riguarda i risultati ottenuti in funzione della molteplicità, risulta evidente che l'andamento dei residui totali non segua una distribuzione gaussiana, mentre i residui relativi a eventi con molteplicità simile (fra 3 e 5 o fra 45 e 55) lo fanno.

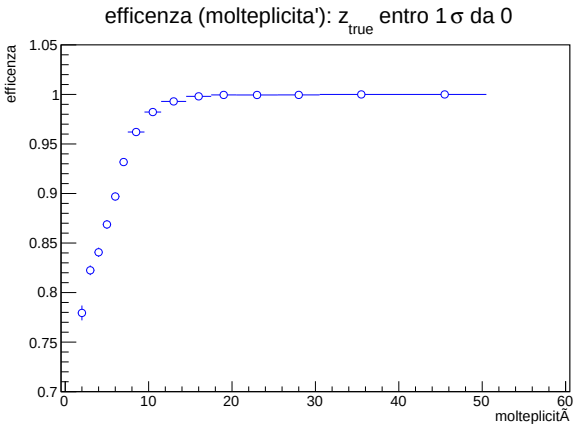


Sono stati anche analizzati gli istogrammi dei residui bin per bin utilizzati per ricostruire la risoluzione in funzione della molteplicità, che è valutata come l'RMS di tali istogrammi. Sono tutti centrati in 0 e con una larghezza di $100 - 400\mu m$, come atteso da un valido algoritmo di ricostruzione.

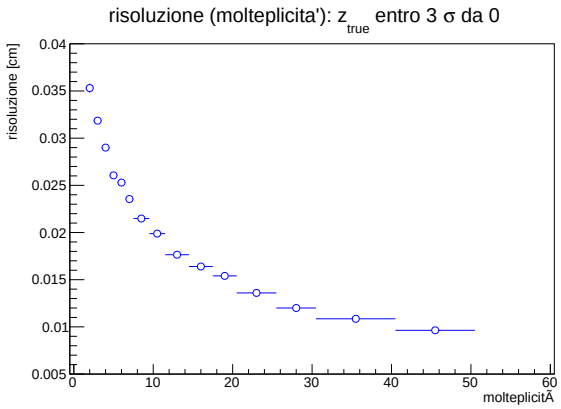


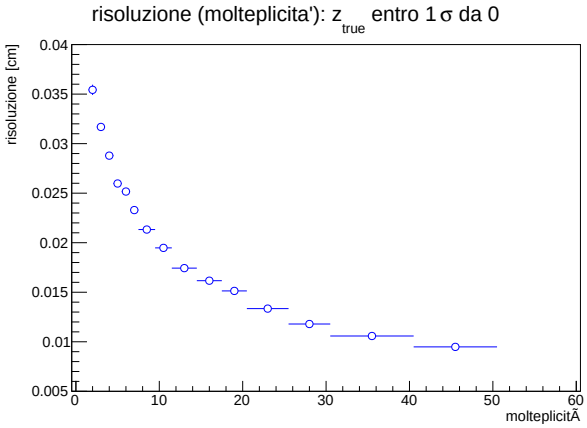
Per quato riguarda l'efficenza, considerando un valore di z_{true} entro 3σ da 0 e entro 1σ da 0 (σ=5.3 cm), si osserva che questa cresce con la molteplicità convergendo a 1 per molteplicità > 15 circa. Eè da notare che prima di tale convergenza la curva per 3σ è leggermente più bassa di quella per 1σ, come atteso dal fatto che la prima contiene anche eventi molto lontani dal centro del rivelatore.



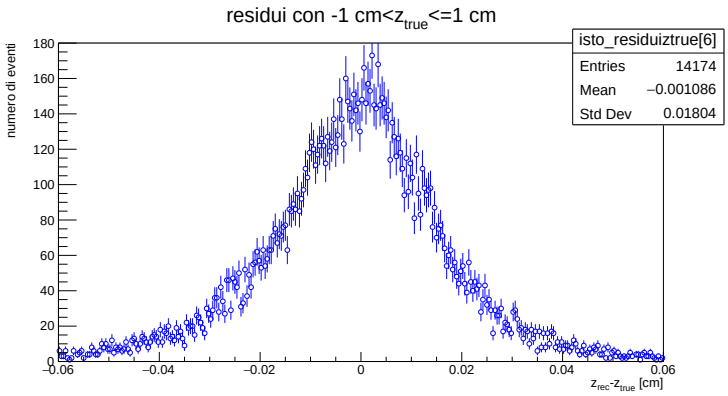


L'andamento della risoluzione è, come atteso, decrescente con la molteplicità (più tracce si hanno meglio si determina il vertice).Per il motivo già citato per l'efficenza, si osserva che la curva per 3σ è un po' più alta (vertice determinato meno bene) rispetto a quella per 1σ.

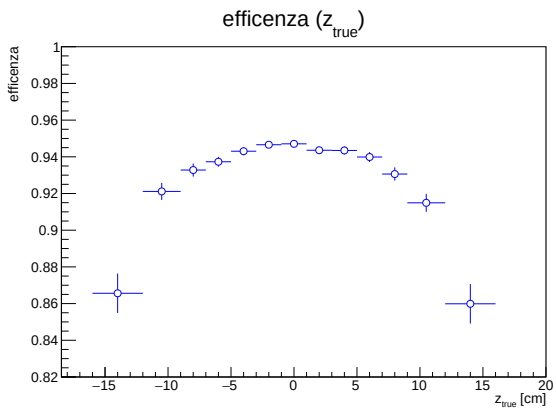


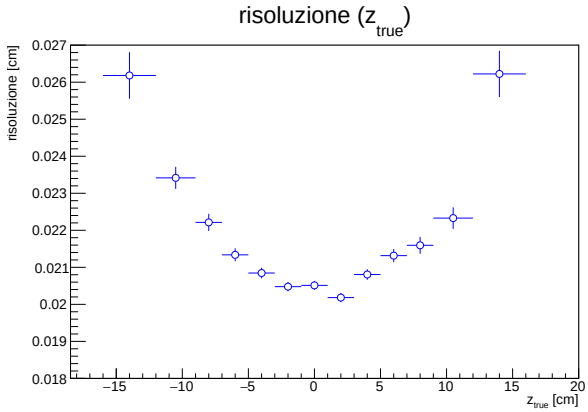


Le stesse analisi sono state nello studio dei risultati in funzione di z_{true}.
Per quanto riguarda i residui, si ottengono andamenti analoghi a quelli ottenuti in funzione della molteplicità.



L' andamento dell'efficienza riporta un massimo in 0 e decresce ai lati (come atteso visto che molti degli eventi generati più esternamente producono tracce che escono dall'apparato). Per la risoluzione risulta l'andamento opposto: si ha il minimo in 0 e l'andamento cresce ai lati.





Le piccole deviazioni osservate rispetto ai risultati attesi si imputano a una bassa statistica (a seguito di rispetute simulazioni).