Simulazione di un rivelatore di vertice

Giulia Italia[1] and Daniel Morris Sulluchuco Huaman[1]

[1]Curriculum di Fisica Nucleare e Sub-Nucleare

20 febbraio 2022

Sommario

Si consideri un rivelatore di vertice costituito da due layer di pixel al silicio coassiali con l'asse del fascio. La beam pipe di berillio avrà raggio 3 cm ed estensione longitudinale infinita sulla scala dell'esperimento. I due layer invece avranno rispettivamente raggio 4 cm e 7 cm ed entrambi estensione longitudinale di 27 cm; si trascuri lo spessore dei rivelatori.

Verranno simulate particelle ad alto momento, in modo tale da poter assumere le loro traiettorie localmente rettilinee. Inoltre sarà possibile generare dei punti scorrelati su entrambi i layer in modo da simulare l'effetto di particelle di basso momento soggette all'azione del campo magnetico ed il rumore del rivelatore.

Lo scopo del rivelatore di vertice è quello di ricostruire la coordinata z del vertice primario di collisione. Si utilizzerà una FAST2 simulation, con smearing gaussiano dei punti di impatto per la ricostruzione

Il programma è suddiviso in due parti: simulazione e ricostruzione. Si riportano di seguito delle brevi descrizioni riguardanti la struttura e la suddivisione in classi.

1 Simulazione

Viene effettuata con la macro **simulation.C** che contiene una funzione principale
*simulation(***TString** *nameFile,* **int** *kNevents,* **bool** *scatt,* **int** *Nnoise)*
la quale prende come argomenti il nome di file di Output, il numero di eventi da generare, una variabile booleana che inizializza lo scattering spento (FALSE) o acceso (TRUE), il numero di punti di noise da generare su ogni layer. I risultati della simulazione vengono salvati nel file .root di output che contiene un TTree i cui rami contengono le coordinate del vertice e la relativa molteplicità e oggetti Hits che rappresentano i punti di collisione sul primo e secondo layer.

In questa macro è possibile modificare i generatori utilizzati per la distribuzione di molteplicità (riga 46) e per la Z del vertice (riga 49).

Si mostrano di seguito le classi implementate che vengono utilizzate nella macro simulation.C e loro principali caratteristiche.

1.1 Hits

Ha come data member: la coordinata z, l'angolo ϕ azimutale ed un indice che può essere utilizzato per verificare che due punti di collisione su layer diversi appartengano alla stessa particella. Ci sono due costruttori: il primo utilizzato nella simulazione per il salvataggio delle informazioni relative al punto di collisione di una particella con uno dei due layer; il secondo verrà utilizzato nella parte di ricostruzione per il salvataggio delle informazioni ottenute dallo smearing.

1.2 MyGen

Classe che eredita da TRandom3 in cui sono stati implementati tutti i generatori pseudocasuali utilizzati nella simulazione:

- Generatori per la posizione del vertice:
  distribuzione normale *genVerX(), genVerY(), genVerZ()*;
  distribuzione uniforme in Z tra [-30,30] cm *genUnifVerZ()* ;

- Generatori di molteplicità :
  da distribuzione assegnata *genDistrMult()*;
  distribuzione uniforme *genUnifMult()*;
  a valore costante *genFixMult()*;

- Generatori della direzione della particella:
  per l'angolo azimutale ϕ con distribuzione uniforme tra [0,2π] *genPhi()*;
  per l'angolo θ della direzione iniziale della particella ricavato da distribuzione assegnata in pseudorapidità *genThetaDistr()*;
  per l'angolo θ per multiple scattering da distribuzione normale *genThetaScatt()*;

- Generazione dei punti di noise *genNoise()* individuati da coordinata Z con distribuzione uniforme tra [-13.5,13.5] cm e angolo azimutale ϕ con distribuzione uniforme tra [0, 2π];

Il costruttore prende come argomenti il seed ed un intero che fissa il valore massimo di molteplicità uniforme. La funzione privata *InHisto()* serve ad inizializzare gli istogrammi ricavati dal file "kinem.root" utilizzati per le distribuzioni assegnate di molteplicità e pseudorapidità.

1.3 PCollision

Oggetto utilizzato per il calcolo dei punti di intersezione tra la traiettoria della particella e beam pipe/ layer 1/ layer 2.

La funzione *SetPar(***double** *x0,* **double** *y0,* **double** *z0 ,* **double** *theta0 ,* **double** *phi0)*
resetta i parametri prima di una nuova collisione e inizializza le coordinate del punto di partenza e la direzione iniziale della particella.

La funzione *CalcPIntersection (***double** *r)* calcola il punto di intersezione della traiettoria della particella con la superficie cilindrica di raggio r; inoltre nel caso in cui la collisione avvenga con uno dei due rivelatori viene effettuato un controllo tramite la funzione privata *IntersectionCheck (***double** *r)*, la quale verifica che il punto di intersezione trovato sia effettivamente sulla superficie cilindrica di raggio r e lunghezza pari all'estensione longitudinale del rivelatore.

1.4 Scattering

Oggetto utilizzato per il calcolo del multiple scattering. Il costruttore prende come argomento una variabile booleana che si comporta come un interruttore (**TRUE=ON**).

La funzione *SetInitialPars(***double** *theta0,* **double** *phi0)* resetta i parametri prima dello scattering e inizializza la direzione iniziale della particella nel S.R. del laboratorio (Oxyz).

La funzione *Rotate(***double** *RndmTheta,* **double** *RndmPhi,* **bool** *checkCol)* prende come argomenti gli angoli generati casualmente (si utilizzano le funzioni implementate nella classe MyGen) che rappresentano la nuova direzione della particella nel S.R. (O'x'y'z') e una variabile booleana che indica se precedentemente è avvenuta una collisione. Se lo scattering è ON ed è avvenuta una collisione allora calcola la nuova direzione della particela nel S.R. di laboratorio (Oxyz), altrimenti inizializza la nuova direzione uguale a quella iniziale.

La funzione privata *Converter()* trasforma le componenti del versore della nuova direzione della particella nel S.R. di laboratorio (Oxyz) nei corrispettivi θ e ϕ.

2 Ricostruzione

Viene effettuata con la macro **recostruction.C** la quale contiene una funzione *reconstruction(***TString** *nameFile)* che prende come argomento il nome del file root ottenuto come output dalla simulazione che si vuole utilizzare per effettuare la ricostruzione.

Per ogni evento, si effettua lo smearing gaussiano dei punti di collisione precedentemente trovati sui due layer. Si cercano poi le possibili posizioni della Z del vertice, utilizzando la funzione *TrackletsRec(..)* per i punti ricostruiti sui due layer che si trovano entro una certa differenza di angolo azimutale. Tutti questi valori vengono salvati su un vector (che viene svuotato alla fine di ogni evento) e usati per riempire un istogramma (anche questo svuotato alla fine di ogni evento); gli elementi del vettore vengono messi in ordine crescente.

Per trovare la Z ricostruita vengono analizzati diversi casi:

1) Il vector è vuoto:
   non si riesce a ricostruire (in questo caso si pone Zrec = 200 cm).

2) La frequenza massima dell'istogramma è = 1

   a) *Se il vector contiene un solo elemento:* la Z ricostruita è pari al valore di quel singolo elemento.

   b) *Se il vector contiene più di un elemento:* si cerca il gruppo con il numero massimo di elementi che si trovano entro una certa distanza (1 mm) dal primo valore considerato; se alla fine si trova più di un gruppo con lo stesso numero di elementi non si ricostruisce; altrimenti la Z ricostruita è pari alla media degli elementi del gruppo.

3) La frequenza massima dell'istogramma è >1:

   a) *Se si trova un unico picco con frequenza massima:* si fa la media dei valori che si trovano in un intorno di ampiezza 1mm del centroide del bin avente picco massimo;

   b) *Se trovo più picchi aventi stessa frequenza massima:* se i centroidi dei bin corrispondenti a tali picchi si trovano entro una certa distanza (1mm) allora si calcola la media dei punti che si trovano in un intorno di 2mm centrato nel valor medio tra i due centroidi; se la distanza tra i centroidi dei due picchi è maggiore del valore fissato allora non si ricostruisce.

I risultati vengono salvati in un file .root che contiene una ntupla avente come rami la coordinata Z del vertice ottenuto dalla ricostruzione, la Z generata nella simulazione e la molteplicità.

Si illustrano di seguito le classi implementate utilizzate nella macro **reconstruction.C** e loro principali caratteristiche.

2.1 Hits

E' la stessa classe utilizzata per la simulazione illustrata precedentemente.

2.2 TrackRec

Classe che eredita da TRandom3 che viene utilizzata per effettuare lo smearing gaussiano sulla coordinata Z e sull'angolo ϕ dei punti di intersezione con il

rivelatore tramite le funzioni *genZrec(), genPhirec()*.

La funzione *TrackletsRec(***double** *x1,* **double** *y1,* **double** *x2,* **double** *y2)* trova l'equazione di una retta passante per due Hits appartenenti a layer diversi e restituisce la coordinata Z ottenuta come intersezione di tale retta con l'asse del fascio.

3 Analisi

Le macro utilizzate per realizzare i grafici sono:

moltepl.C e **ztrue.C** che contengono ciascuna una funzione che prende come argomento il nome del file root ottenuto come output dalla ricostruzione; rispettivamente *moltepl(***TString** *nameFile)* e *ztrue(***TString** *nameFile)*

moltepl.C realizza i grafici di efficienza vs molteplicità e risoluzione vs molteplicità. **ztrue.C** realizza i grafici di efficienza vs zTrue e risoluzione vs zTrue.

In entrambe le macro l'errore sull'efficienza è stato calcolato usando la seguente formula per la varianza ricavata e giustificata nel paper citato nei riferimenti.

$$V(\epsilon) = \frac{(k+1)(k+2)}{(n+2)(n+3)} - \frac{(k+1)^2}{(n+2)^2}$$

dove n è il numero di vertici generati e k è il numero di vertici ricostruiti.

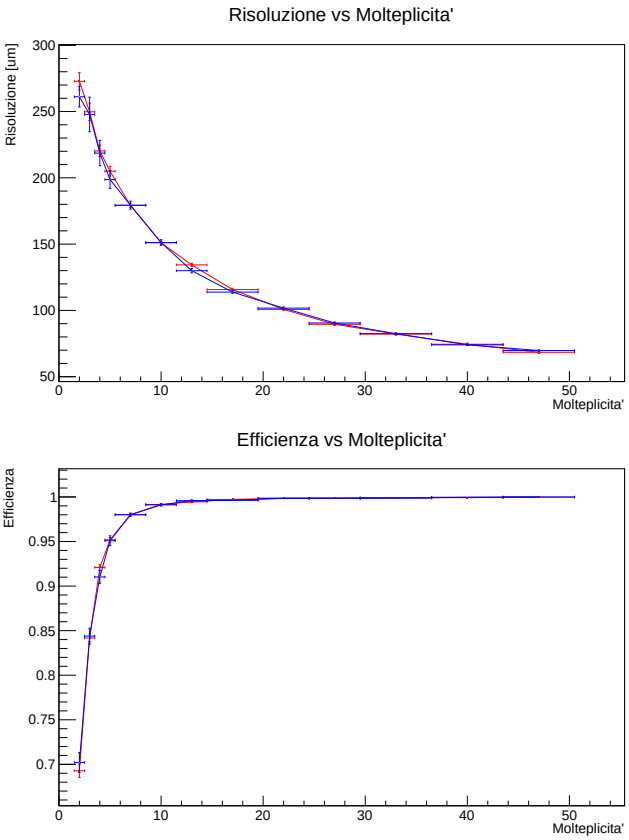

Figura 1: Rosso molt. assegnata - Blue uniforme Zvertice entro 3 sigma

4 Esecuzione del programma

Passi per eseguire la simulazione e ricostruzione:

1) .x compclass.C
2) simulation("OFile",nEvents,Scattering,nNoise);
   es. simulation("sim",100000,1,20);
3) reconstruction("NomeFileOutputSimulazione");
   es. reconstruction("sim");

Per realizzare i grafici:

a.1) .L moltepl.C
a.2) moltepl("NomeFileOutputRicostruzione");
   es. moltepl("rec_sim");
b.1) .L ztrue.C
b.2) ztrue("NomeFileOutputRicostruzione");
   es. ztrue("rec_sim");

Risultati

Si riportano alcuni grafici ottenuti in seguito ad una ricostruzione relativa ad una simulazione con 100'000 eventi, Scattering ON e 20 punti di noise per layer.

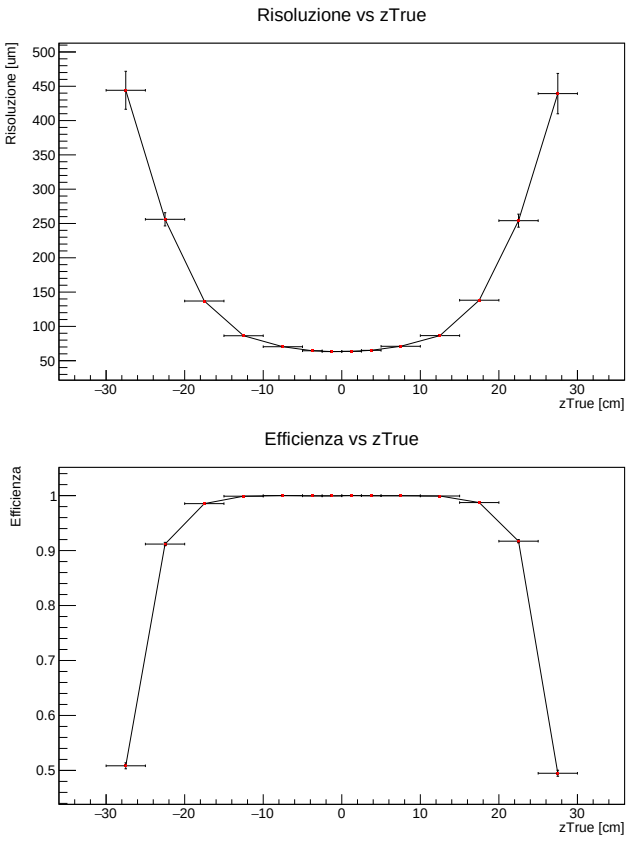

Figura 2: Molteplicità 50 e Zvertice uniforme

Riferimenti bibliografici

[1] T. Ullrich and Z. Xu : Treatment of Errors in Efficiency Calculations. Brookhaven National Laboratory