Progetto rilevatore

Giacomo Scali
Alessio Alvino

July 23, 2024

1 Introduzione

L'obbiettivo del codice è quello di simulare un rilevatore di raggi cosmici con forma e dimensione scelta in input dall'utente sotto le seguenti ipotesi:

- 1 particella per cm^2 al minuto.
- Numero di particelle nell'intervallo di tempo distribuito secondo Poisson.
- Particelle distribuite uniformemente nell'intervallo di tempo.

2 Organizzazione del codice

Il codice è organizzato nelle seguenti classi: Figura, Rettangolo, Cerchio, Triangolo e Rilevatore; con Rettangolo, Cerchio, Triangolo che sono classi figlie di Figura.
Ogni una delle classi figlie di Figura possiede un metodo "Disegna" e un metodo "Area", programmati come polimorfismi dei metodi "Disegna" e "Area" di Figura, che: disegnano a schermo la geometria del rivelatore e ne calcolano l'area.
Abbiamo anche inserito una classe "Rilevatore" con i metodi: "questions" che crea la figura secondo un input dell'utente, "distribuzizoneNP" che genera i dati relativi al numero di particelle in ogni intervallo di integrazione e realizza un fit sull'istogramma; "distribuzioneT" che calcola i dati relativi ai tempi di arrivo delle particelle e realizza un fit sull'istogramma.

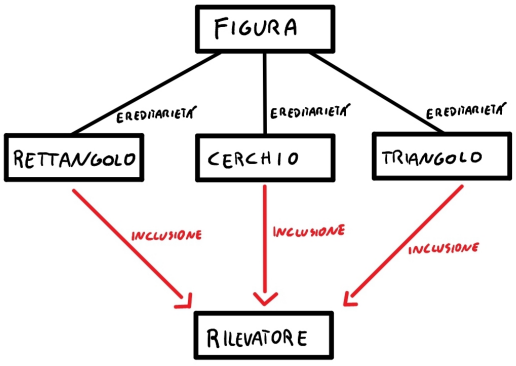

Figure 1: Organizzazione classi

Proseguiamo seguendo una esecuzione del codice:

```
//Main_2
int main(int argc, char* argv[]){
TApplication app("app", &argc, argv);
TCanvas* c = new TCanvas("canvas", "Forma Rilevatore e distribuzione",
0, 0, 1600, 1200);

Rilevatore rl(c);
rl.questions();
rl.distribuzioneNP();
rl.distribuzioneT();

c->Modified(); c->Update();

TRootCanvas *rc = (TRootCanvas *) c->GetCanvasImp();
rc->Connect("CloseWindow()", "TApplication", gApplication,
"Terminate()");

app.Run();
return 0;
}
```

TApplication: Crea un oggetto di tipo TApplication, che rimane "attivo" anche al termine dell'esecuzione del programma e consente di tenere aperta la finestra grafica creata successivamente. Tcanvas* crea un puntatore a un oggetto di tipo Tcanvas, allocato dinamicamente, di nome c, che viene successivamente diviso in 4 parti.

Rilevatore rl(c): Crea un oggetto appartenente alla classe rilevatore chiamato rl.

rl.questions(): Chiama il metodo questions della classe rilevatore che permette l'inserimento dell'intervallo di integrazione, tempo totale di acquisizione (in minuti) e la selezione della geometria del rilevatore con una codifica "Rettangolo=0, Triangolo=1, Cerchio=2"; crea poi la relativa figura, il quale costruttore chiede le dimensioni necessarie al calcolo dell'area (in cm).

```
void Rilevatore::questions()
  {
    cout << "Intervallo di integrazione (in minuti): ";
    cin >> integ;
    cout << "Tempo totale di acquisizione (in minuti): ";
    cin >> runtime;
    runtime= runtime;
    cout << "Forma del rilevatore (possibilita: Rettangolo=0,
        Triangolo=1, Cerchio=2): ";
    cin >> fig;
```

```
if (fig==0)
  Pt = new Rettangolo(r);
if (fig==1)
   Pt = new Triangolo(r);
 if (fig==2)
 Pt = new Cerchio(r);
 Pt->Disegna();
 Ar= Pt->Area();//prende l'area e la mette dentro Ar
 }
```

rl.distribuzioneNP(): Chiamata al metodo di rilevatore che genera il numero di particelle all'interno di ogni intervallo di integrazione, utilizzando la libreria "TRandom3.h", estraendo da una distribuzione di Poisson con:
λ =numero_atteso_particelle per cm^2al minuto $\cdot$ Area_rilevatore $\cdot$ Intervallo_integ.
Effettua poi l'inserimento dei valori nell'istogramma [1] e il fit della distribuzione di Poisson.

```
// void Rilevatore::distribuzioneNP()
.
poisson_distribution<int> pd(Ar*integ);
for (int i = 0; i < runtime/integ; ++i)
{
vec_i.push_back(pd(gen));
}
.
.
// Riga 56-73 Rilevatore.cpp
// Istogramma e fit

r->cd(2);

TH1F* h = new TH1F("h", "Grafico del numero di particelle",
Ar*1*integ*2, 0, Ar*1*integ*2);

for (int i = 0; i < runtime/integ ; ++i){
h->Fill(vec_i[i]);
}
h->Draw();
.
.
h->Fit( fPoissonByHand,"","R",0,Ar*integ*2)
```

[1] Numero di bin ed estremi del grafico si adattano automaticamente

`rl.distribuzioneT()`

Chiamata al metodo di rilevatore che calcola il numero totale di particelle registrate e genera i tempi di arrivo per ogni particella estraendo da una distribuzione uniforme su tutta la durata dell'esperimento (run time); i tempi ottenuti vengono inseriti in un vettore (poi ordinato in ordine crescente) dal quale calcoliamo gli intertempi da inserire nell'istogramma.
Inserisce poi i valori nell'istogramma e esegue il fit della distribuzione esponenziale con:
λ =Area_rilevatore $\cdot$ numero_atteso_particelle_per cm^2 al minuto.

```
void Rilevatore::distribuzioneT()


{
// Generazione tempi arrivo
   vector<float> vec_f;
   int NP_tot;
   random_device rand_dev;
   mt19937 generator(rand_dev());
   uniform_real_distribution<float> distr(0, runtime);
   for (int i=0; i< runtime/integ; i++)
      {
   NP_tot = NP_tot + vec_i[i];
      }

   for (int i = 0; i < NP_tot; ++i){
       vec_f.push_back(distr(generator));
    }

// Ordinamento del vettore
    float j;
    for (int i=0; i< NP_tot-1; i++)
    {
      for (int k=i+1; k < NP_tot; k++)
         {
           if (vec_f[i]>vec_f[k]){
            j=vec_f[k];
            vec_f[k]=vec_f[i];
            vec_f[i] = j;
            }
         }
    }
```

```
// Calcolo degli intertempi
vector<float> vec_T;
    for (int i=0; i < NP_tot-1; i++)
      {
   vec_T.push_back(vec_f[i+1]-vec_f[i]);
      }

// Disegno isogramma e fit
    r->cd(3);

    TH1F* h2 = new TH1F("h", "Grafico della distribuzione dei tempi di
        arrivo", sqrt(NP_tot-1)/3, 0, 1/Ar*3);

for (int i = 0; i < NP_tot-1 ; ++i){
    h2->Fill(vec_T[i]);
}
```

3 Risultati ottenuti

Effettuando un test con un rilevatore di area = 100 cm^2, tempo di integrazione = 5 s ed un run time di 800 minuti otteniamo i seguenti risultati:

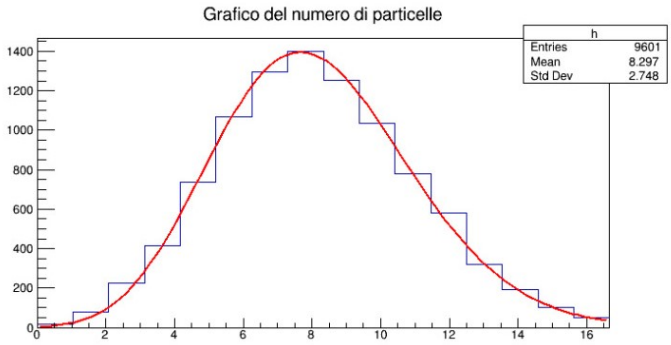

Figure 2: Istogramma numero particelle caso 1

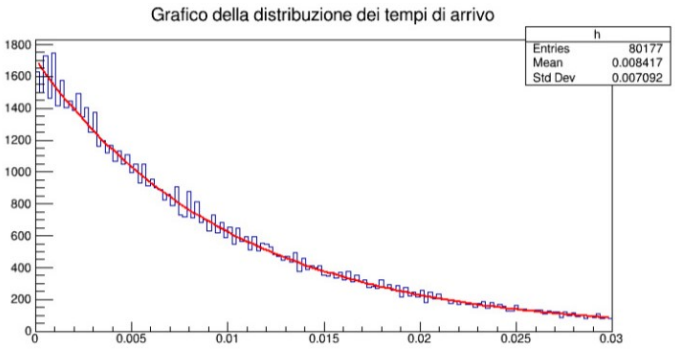

Figure 3: Istogramma intertempi caso 1

Effettuando un test con un rivelatore con area = 1 m^2, tempo di integrazione = 1 s ed un run time di 10 minuti otteniamo i seguenti risultati.

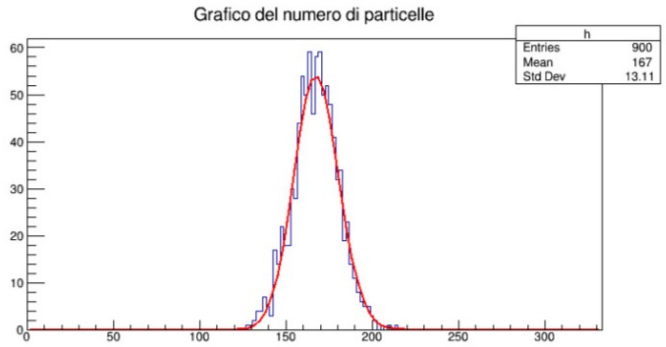

Figure 4: Istogramma numero particelle caso 2

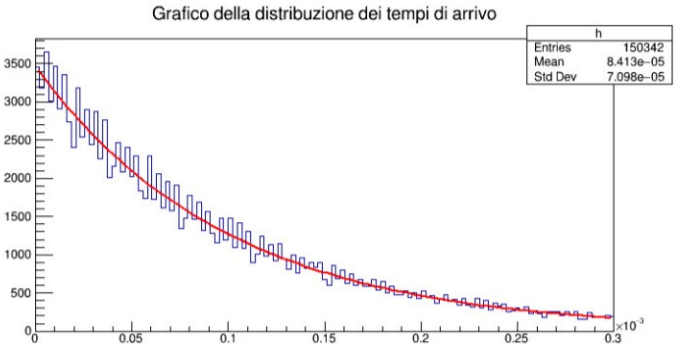

Figure 5: Istogramma intertempi caso 2

In entrambi i casi abbiamo ottenuto istogrammi che sembrano rispecchiare quanto ci si aspetta dalle rispettive distribuzioni. Infatti l'istogramma del numero di particelle per ogni intervallo di integrazione deve seguire una distribuzione di Poisson, dato che esprime le probabilità per il numero di eventi che si verificano successivamente ed indipendentemente in un dato intervallo di tempo (inoltre è proprio la distribuzione utilizzata per generare i dati).
Per gli intervalli tra i tempi di arrivo delle particelle invece la distribuzione attesa risulta essere una esponenziale, infatti è risultato noto [Gallager, 1996] che i tempi di arrivo per un processo di Poisson seguano una distribuzione esponenziale.

References

[Gallager, 1996] Gallager, R. G. (1996). *Poisson Processes*, pages 31–55. Springer US, Boston, MA.