Metodologie di Analisi Dati

Andrea Sala

Problema 1

Esercizio 1

Supponiamo di avere $r_i \in (0, 1]$ variabili casuali uniformemente distribuite. Costruire istogrammi delle seguenti variabili:

(a) $x = r_1 + r_2 - 1$

(b) $x = \sum_{i=1}^{4} r_i - 2$

(c) $x = \sum_{i=1}^{12} r_i - 6$

Gli istogrammi sono stati costruiti estraendo, per un numero di volte pari a `throws` $= 10^5$, 12 numeri casuali, e aggiornando opportune variabili di somma con tutte queste variabili (nel caso di c), solo quelle divisibili per 3 (nel caso di b) e solo quelle divisibili per 6 (nel caso di a). Da ultimo, ho completato le variabili sottraendo il numero intero richiesto e ho riempito i tre istogrammi.

```
for (int i=0; i<throws; i++){
      double sum2=0.;
      double sum4=0.;
      double sum12=0.;
      for (int y=1; y<13; y++){
          rnd = ran->Rndm();
          r.push_back(rnd);
          if (y%6==0) sum2+= rnd;
          if (y%3==0) sum4+= rnd;
          sum12 +=rnd;
      }
    h1a->Fill(sum2 - 1.0);
    h1b->Fill(sum4 - 2.0);
    h1c->Fill(sum12 - 6.0);
  }
```

La restante parte del codice è dedicata al layout degli istogrammi e alla creazione delle canvas. Le medie delle variabili definite sopra si ricavano facilmente in quanto $E[r_i] = \frac{1}{2}$, dunque in tutti e tre i casi si ha $E[x] = 0$. Le varianze si ricavano in modo esatto conoscendo

$$\sigma^2(r_i) = E[r_i^2] - (E[r_i])^2 = \int_{-\infty}^{+\infty} r_i^2 f(r_i) dr_i - \left(\int_{-\infty}^{+\infty} r_i f(r_i) dr_i \right)^2 = \frac{1}{12} \tag{1}$$

Poiché le r_i non sono correlate tra di loro, la varianza della somma di variabili è semplicemente la somma della varianze:

(a) $x = r_1 + r_2 - 1 \quad \Rightarrow \quad \sigma_x^2 = \frac{1}{6}$

(b) $x = \sum_{i=1}^{4} r_i - 2 \quad \Rightarrow \quad \sigma_x^2 = \frac{1}{3}$

(c) $x = \sum_{i=1}^{12} r_i - 6 \quad \Rightarrow \quad \sigma_x^2 = 1$

I risultati ottenuti con gli istogrammi sono mostrati in figura 1.
Nella Tabella 1 sono confrontati i valori esatti delle deviazioni standard con quelli ottenuti dagli istogrammi. L'accordo è molto buono.

	Esatto (frac)	Esatto (dec)	Campionato
(a)	$1/\sqrt{6}$	0.4082	0.4085
(b)	$1/\sqrt{3}$	0.5774	0.5763
(c)	1	1.000	1.001

Tabella 1: Confronto deviazioni standard.

Questo esercizio inoltre mostra il funzionamento del Teorema del Limite Centrale. Si osserva che più il numero di variabili casuali aumenta, più la distribuzione della grandezza $\frac{x-\mu}{\sigma}$ tende

alla distribuzione normale standard. Nel caso di distribuzione uniforme, le ipotesi del Teorema sono rispettate e dunque si osserva una convergenza verso un profilo gaussiano. A certificazione di questo, ho eseguito un fit di ML con una curva gaussiana nel caso (c), ottenendo un risultato discreto. In realtà, basterebbe aumentare il numero di bin e di conseguenza il numero di gradi di libertà er far abbassare ulteriormente il valore di χ^2/ndf.

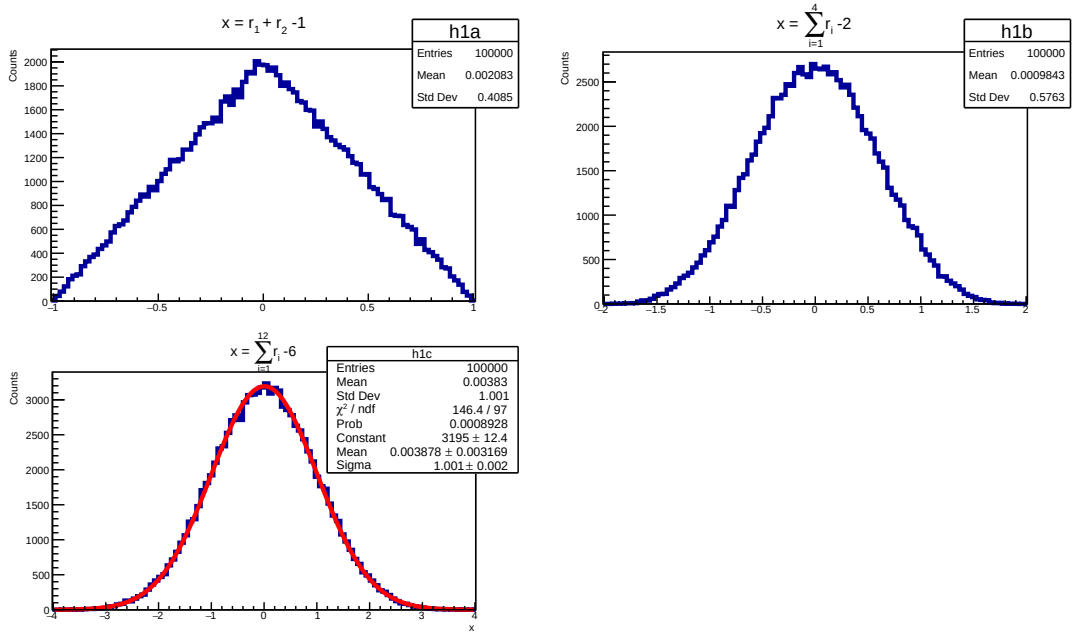


Figura 1: Istogrammi di variabili casuali.

Esercizio 2

Considerando una pdf a "dente di sega" ($x_{max} = 1$):

$$f(x) = \begin{cases} \frac{2x}{x^2_{max}} & 0 < x \leq x_{max} \\ 0 & \text{altrimenti} \end{cases} \tag{2}$$

Generare numeri casuali distribuiti secondo $f(x)$ utilizzando dapprima il metodo della trasformazione inversa, e poi un metodo *accept-reject*.
Il metodo della trasformazione inversa consiste nell'invertire la funzione cumulativa, in questo caso

$$y = F(x) = \int_{-\infty}^{x} f(x')dx' = \frac{x^2}{x^2_{max}} \qquad \text{se } 0 < x \leq x_{max} \tag{3}$$

Da cui si ricava

$$x = \sqrt{y} \cdot x_{max} \tag{4}$$

dove $y \in (0, 1]$ è un numero casuale estratto da una distribuzione uniforme.
Il codice è molto semplice, basta riempire un istogramma con la quantità x definita in 4.

```
for (int i=0; i<throws; i++){

        r1 = ran->Rndm();
        h2a->Fill(pow(r1,0.5)*xmax);

}
```

L'istogramma dei risultati è mostrato in Figura 2.
La seconda parte dell'esercizio consiste invece nell'implementazione di un metodo *accept-reject*. Questo è stato fatto estraendo coppie di numeri casuali distribuiti uniformemente in $(0, x_{max})$ e in $(0, \max f(x))$ rispettivamente. A quel punto, viene fatto il controllo se il secondo numero è minore della funzione valutata nel primo numero. In caso affermativo il primo numero viene accettato e va a riempire l'istogramma.

```
  for (int i=0; i<throws; i++){
        r2 = ran->Rndm();
        r3 = ran->Rndm()*fmax;
        if (r3<f(r2,xmax)) h2b->Fill(r2);
    }
    ...
  double f( double x, double xmax){
    if (x>0 && x<=xmax) return 2*x/pow(xmax,2);
    else return 0;
    }
```

Il risultato è mostrato in Figura 2. Si osserva che le distribuzioni ricavate con i due diversi metodi vanno a coincidere.

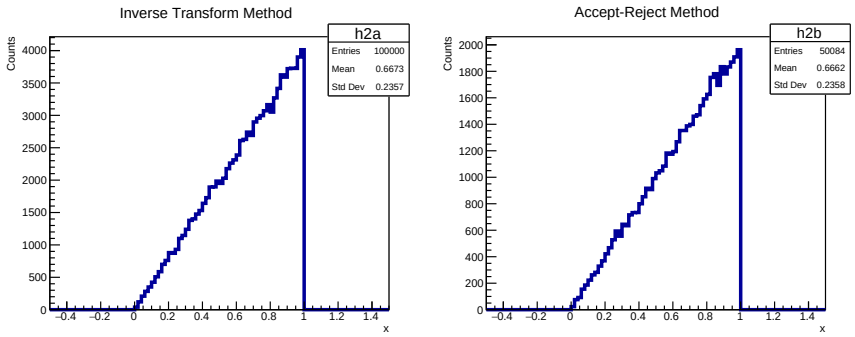


Figura 2: Pdf a "dente di sega".

Problema 2

Esercizio 1

Il Problema 2 riguarda l'utilizzo del pacchetto TMVA per svolgere una semplice analisi di classificazione segnale-fondo con dei dati forniti dal problema. Per prima cosa, viene richiesto di allenare e testare un classificatore lineare come il discriminante di Fisher. Al termine della fase di allenamento, il programma produce un file nella directory `weights` contente i pesi ottimali per massimizzare la distanza tra le due distribuzioni di segnale e background. Il discriminante di Fisher è infatti definito come

$$t(\vec{x}) = w_0 + \sum_{i=1}^{3} w_i x_i \tag{5}$$

e si cerca di trovare il vettore di pesi $\vec{w}$ tale per cui è massimizzata la distanza tra le distribuzioni

$$J(\vec{w}) = \frac{(E[t|s] - E[t|b])^2}{V[t|s] + V[t|b]} \tag{6}$$

Questo classificatore fornisce un *boundary* lineare per separare le due classi. In questo problema vedremo quanto è buona questa approssimazione di separazione lineare tra segnale e fondo.
Supponendo di considerare come eventi di segnale tutti gli eventi con $t_{Fisher} > 0$, ho calcolato l'efficienza del segnale e del fondo $\varepsilon_{s,b}$ nonché la purezza del segnale p_s, supponendo che le probabilità a priori di ottenere fondo e segnale siano identiche. Queste quantità sono così definite:

$$\varepsilon_{s(b)} = P(t_{Fisher} > 0 | s(b)) \qquad p_s = P(s | t_{Fisher} > 0) \tag{7}$$

Per fare questo ho implementato all'interno del programma di analisi le seguenti righe di codice:

```
1
2  double tCutFisher = 0.0;
3  double tFisher = reader->EvaluateMVA("Fisher")
4
5  if (i==0) { \\signal
6
7      if (tFisher>tCutFisher) {
8              FSigEff++;
9              tGrt0++;
10     }
11     hFishSig->Fill(tFisher);
12 }
13 if (i==1) { \\background
14
15     if (tFisher>tCutFisher) {
16             FSigEff++;
17             tGrt0++;
18     }
19     hFishBkg->Fill(tFisher);
20 }
21
22 ...
23
24 std::cout << ">>>> \t Fisher Signal efficiency: " << (double)FSigEff/nSig ;
25 std::cout << ">>>> \t Fisher Background efficiency: " << (double)FBkgEff/nBkg ;
26 std::cout << ">>>> \t Fisher Signal purity: " << (double)FSigEff/tGrt0 ;
```

La variabile `tGrt0` è un accumulatore per contare tutti gli eventi che hanno $t > 0$, in modo da poter ottenere la purezza del segnale. Inoltre, nel codice sono inclusi anche i comandi per riempire gli istogrammi di t_{Fisher} di fondo e di segnale. Questi istogrammi sono mostrati in Fig. 3, mentre i risultati numerici di efficienza e purezza sono riportati in Tabella 2. Osservando i valori in tabella si nota che il discriminante di Fisher è un buon classificatore (soprattutto per essere un classificatore lineare), in quanto riesce a separare il campione in due zone, riuscendo a isolare più del 95% di segnale nella zona $t > 0$; all'interno di questa zona, solamente l'8% degli eventi appartiene alla distribuzione di fondo.
Successivamente, ho implementato all'interno del codice in modo da includere un classificatore non lineare, in particolare un Multi Layer Perceptron (MLP), ovvero una rete neurale artificiale. Questo classificatore va a minimizzare una *loss function* opportunamente definita e produce,

come nel caso precedente, un insieme di pesi salvati in un file. Dopo aver invocato il metodo sia nella fase di allenamento che nella fase di analisi, ho riempito due istogrammi con le distribuzioni di segnale e di fondo per la variabile t_{MLP}. Il codice non è riportato perché è formalmente identico a quello del punto precedente. Successivamente, ho calcolato i coefficienti di efficienza e purezza in modo da poter confrontare le performance di MLP con quelle di Fisher. Analizzando i grafici ottenuti, si osserva che il classificatore non lineare riesce a separare in modo molto più netto le due distribuzioni , dimostrando di essere un ottimo classificatore (il tempo necessario al training è però maggiore che nel caso precedente). I coefficienti di efficienza e purezza sono simili a quelli ottenuti con Fisher, ma leggermente migliori (vedi Tabella 2).
N.B. I valori di soglia nei due casi sono differenti ($t_{\text{cut}} = 0$ per Fisher e $t_{\text{cut}} = 0.5$ per MLP) solamente perchè il primo classificatore fornisce in output un numero reale a valori in $(-10, 10)$, mentre MLP fornisce un numero a valori in $(0, 1)$. Il valore di soglia è sempre stato scelto come punto medio dell'intervallo.

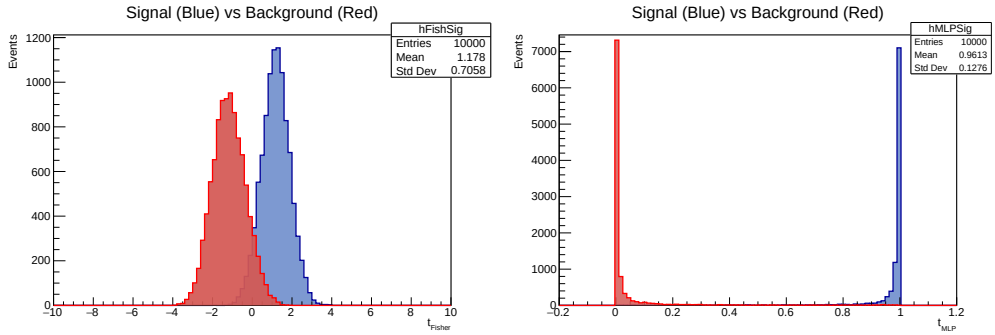


Figura 3: Istogrammi di t_{Fisher} (sinistra) e di t_{MLP} (destra) per le distribuzioni di fondo e segnale.

Metodo	ε_s	ε_b	p_s
Fisher	0.9513	0.0869	0.9163
MLP	0.9766	0.0332	0.9671

Tabella 2: Coefficienti di efficienza e purezza del segnale.

Esercizio 2

Nell'esercizio 2 è richiesta l'implementazione del classificatore Boosted Decision Tree (BDT). Il metodo è stato chiamato secondo la sintassi opportuna e con i parametri richiesti dal problema (processo di boosting con 200 alberi), e sono stati riempiti istogrammi sia per il campione di allenamento che per il campione di test. Il codice usato è uguale a quello usato per chiamare il metodo e riempire gli istogrammi nei punti precedenti.
I risultati sono mostrati in Fig. 4. Si nota che non ci sono sostanziali differenze tra i due campioni: in entrambi i casi, il classificatore riesce a separare piuttosto bene i due istogrammi.

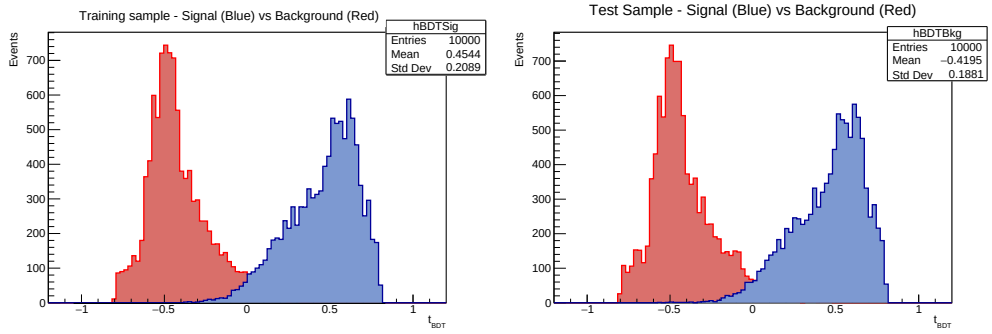


Figura 4: Istogrammi di t_{BDT} del campione di training (sinistra) e di test (destra) per le distribuzioni di fondo e segnale.

Infine, ho ripetuto questa procedura con classificatori BDT con diversi numeri di iterazioni di boosting, in particolare 2,5,10,20,50,100,200,500,1000,2000,5000,1000 e 50000. Ho automatizzato il tutto in un ciclo ricavando l'espressione analitica di tale successione:

$$f(n) = \left([(n-1) \mod 3]^2 + 1 \right) \cdot 10^{\lfloor \frac{n}{3} \rfloor} \tag{8}$$

In fase di training ho invocato un classificatore BDT (cambiandogli nome ogni volta) con un crescente numero di iterazioni:

```
std::string Noftrees, bdtplusn, settings;

for (int i=2; i<14; i++){

  Ntrees = (pow(((i-1)%3),2) + 1 ) * pow(10, int((double)i/3.));
  Noftrees = std::to_string(Ntrees);
  bdtplusn = "BDT_" + Noftrees;
  settings = "H:!V:Ntrees="+Noftrees;
  factory-> BookMethod(dataloader, TMVA::Types::kBDT, bdtplusn, settings);

}
```

In fase di analisi, per ogni classificatore ho calcolato il tasso di errore. Posta la soglia a $t_{BDT} = 0$ in quanto BDT produce numeri in $(-1, 1)$, ho contato quanti eventi di segnale stavano a sinistra della soglia e quanti eventi di background stavano a destra della soglia. All'interno di ogni iterazione ho implementato un controllo del tipo

```
int ErrRate=0;

if (i==0){ //signal

  if(tBDT<0) ErrRate++;

}

if (i==1){ //background

  if(tBDT>0) ErrRate++;

}
```

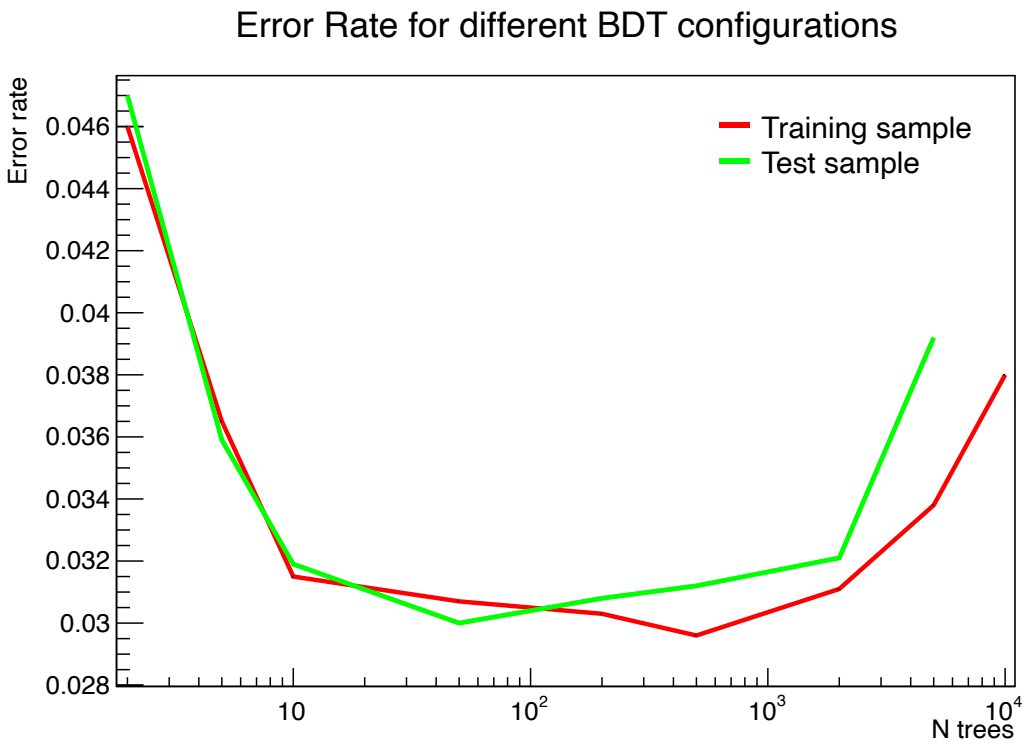


Figura 5: Tasso di errore in funzione del numero di iterazioni

```
...

ErrRate = (double)ErrRate / (nSig + nBkg);
```

Ho fatto un plot del tasso di errore in funzione del numero di iterazioni (in scala logaritmica). Questo grafico, mostrato in figura 5, presenta un largo minimo nella zona $10^1 - 10^3$ iterazioni. Si deduce quindi che il numero ottimale di iterazioni di boosting per questo problema specifico è approssimativamente 500 iterazioni.

Problema 3

Esercizio 1

Questo esercizio fornisce un'introduzione alla classe di ROOT `TMinuit`, utilizzata per minimizzare una data funzione parametrica. Inizialmente, ho eseguito i codici forniti per generare numeri distribuiti secondo

$$f(x;\xi) = \frac{1}{\xi} e^{-\frac{x}{\xi}} \qquad x \geq 0 \tag{9}$$

salvando i dati in un file esterno. Successivamente, ho utilizzato questi dati per eseguire un fit e ottenere una stima dell'unico parametro ξ, chiamata $\hat{\xi}$, con la sua deviazione standard $\sigma_{\hat{\xi}}$. I risultati numerici sono riportati in Tabella 3
In secondo luogo, ho modificato il codice di `makeData` in modo da generare numeri distribuiti secondo la somma di due esponenziali

$$f(x;\xi_1,\xi_2,\alpha) = \frac{\alpha}{\xi_1} e^{-\frac{x}{\xi_1}} + \frac{1-\alpha}{\xi_2} e^{-\frac{x}{\xi_2}} \qquad x \geq 0 \tag{10}$$

con $\alpha = 0.2$, $\xi_1 = 1.0$, $\xi_2 = 5.0$. Per generare 200 numeri distribuiti secondo l'Eq. 10 ho scritto il seguente codice

```
    for (int i=0; i<numVal; ++i){

        double q = ran->Rndm();
        if (q<alpha){
            double r = ran->Rndm();
            double x = -xi1 * log(r);
            dataFile << x << endl;
        }
        else if (q>=alpha){
            double r = ran->Rndm();
            double x = -xi2 * log(r);
            dataFile << x << endl;
        }
    }
```

A questo punto, ho modificato il programma `expFit` adeguandolo alla nuova pdf. Innanzitutto, ho scritto una funzione che replicasse l'Eq. 10 con valori fissati dei parametri:

```
 double NewExpPdf(double* xPtr, double par[]){
   double x = *xPtr;
   double xi1 = par[0];
   double xi2 = par[1];
   double alpha = par[2];
   double f = 0;

   if (x>=0. && xi1>0. && xi2>0. && alpha>0. && alpha<1.){
       f = alpha/xi1 * exp(-x/xi1) + (1.-alpha) / xi2 * exp(-x/xi2);
   }
  return f;
}
```

e poi ho esteso l'implementazione del fit passando dalla configurazione a un parametro a quella a 3 parametri. Per ogni parametro ho inserito un valore iniziale (vicino al valore vero) e una dimensione del passo pari a un decimo del valore vero. Inoltre, ho ampliato l'intervallo su cui è stato eseguito il fit in modo da includere tutti i numeri generati dal programma precedente.

Parametro	Valore vero	$\hat{\theta}$	$\sigma_{\hat{\theta}}$
ξ	1.00	1.03	0.07
ξ_1	1.00	0.599	0.15
ξ_2	5.00	4.9	0.5
α	0.20	0.21	0.06

Tabella 3: Stime numeriche dei parametri

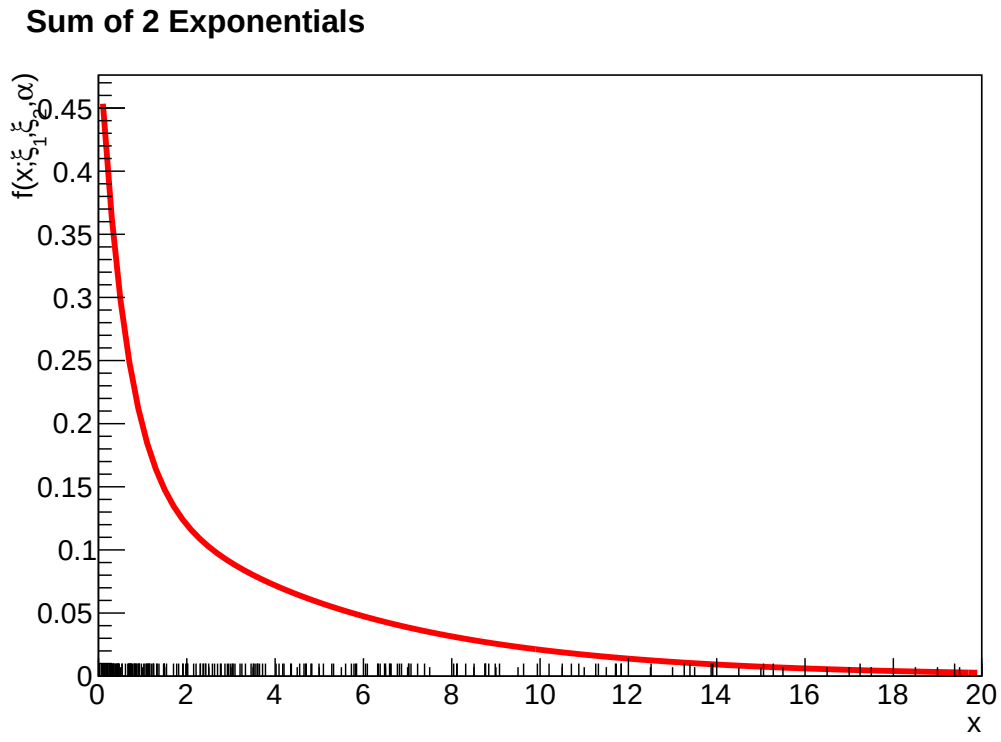


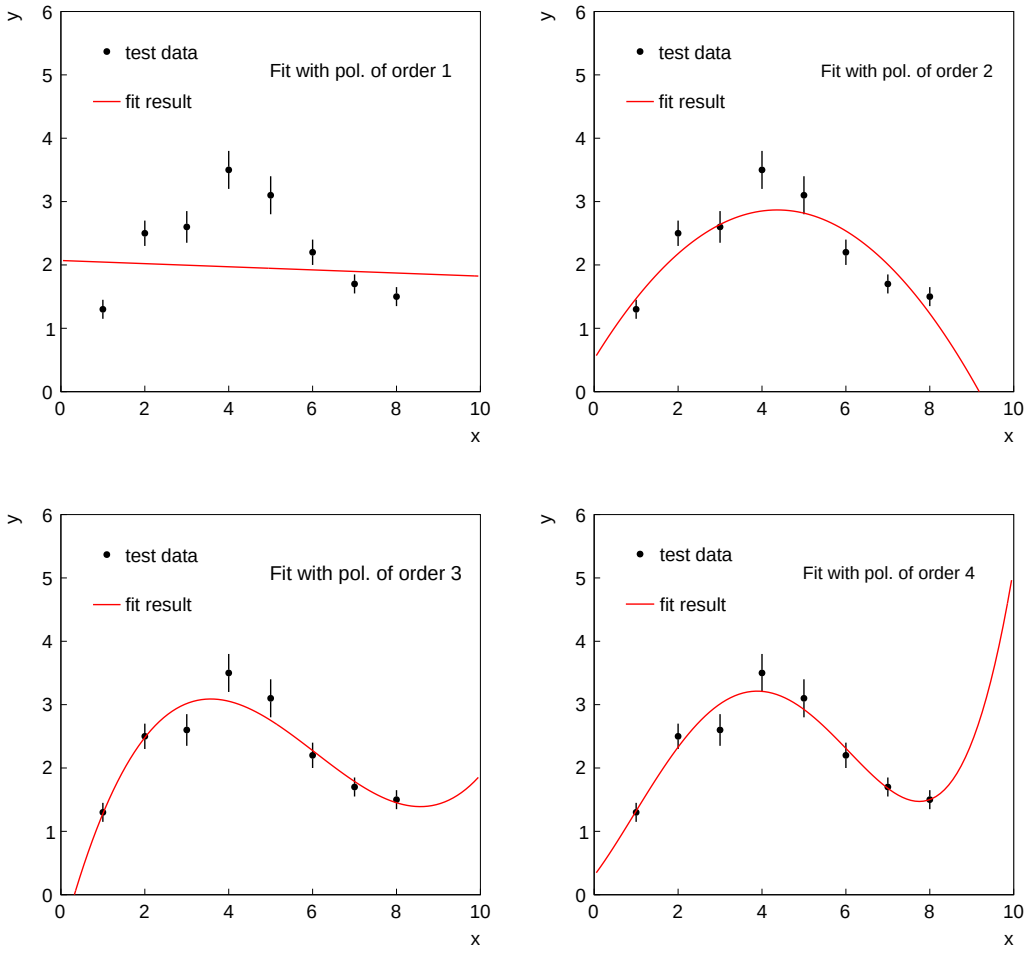
Figura 6: Fit della somma di esponenziali usando Minuit

Nel mio caso, ho applicato il fit per $x \in (0, 20)$. Con l'esecuzione del codice ho ottenuto un plot della funzione fittata, mostrato in rosso nella Fig. 6. I valori numerici per le stime ML dei tre parametri sono riportati in Tabella 3.
Ho inoltre fatto stampare al programma la matrice degli errori (ovvero la matrice di covarianza) e la matrice di correlazione, che vengono riportate di seguito. È importante notare come gli elementi lungo la diagonale della matrice degli errori corrispondano alle varianze, mentre gli elementi diagonali della matrice di covarianza sono sempre pari a 1. Inoltre, entrambe le matrici sono simmetriche per definizione.

$$V_{ij} = \begin{bmatrix} 0.0421 & 0.0343 & 0.00797 \\ 0.0343 & 0.2360 & 0.01740 \\ 0.00797 & 0.01740 & 0.00442 \end{bmatrix} \qquad \rho_{ij} = \begin{bmatrix} 1.000 & 0.344 & 0.584 \\ 0.344 & 1.000 & 0.538 \\ 0.584 & 0.538 & 1.000 \end{bmatrix} \tag{11}$$

Esercizio 2

Questo esercizio fornisce un codice in grado di fittare con il metodo dei minimi quadrati un polinomio di grado regolabile secondo un set di dati presente in un file. In particolare, i dati consistono di 8 terne di numeri (x, y, σ_y). Come prima cosa, ho eseguito il codice variando il numero di parametri liberi e registrando in Tabella 4 i valori di χ^2 e p-value corrispondenti. Si nota che il numero minimo di parametri per avere un p-value maggiore di 0.1 è 4, ovvero un polinomio di terzo grado. Anche nei grafici, riportati sotto, si nota come a partire dal terzo grado i dati vengono fittati con più precisione dal programma.



n_{par}	n	ndf	χ^2	p-value
2	1	6	86.85	$1.36 \cdot 10^{-16}$
3	2	5	19.69	0.0014
4	3	4	6.96	0.13
5	4	3	5.05	0.16

Tabella 4: Risultati del fit.

Il punto successivo chiede di utilizzare la funzione con i parametri "corretti" per ricavare il valore previsto della funzione in $x = 5$, $x = 6$, $x = 10$ per $n = 2, 3, 4$. Questo è stato implementato con

```
for (int i=0; i<npar; i++) f->SetParameter (i, thetaHat[i]);

 const int Nprev = 3;
 double *xprev = new double [Nprev];
 double *yprev = new double [Nprev];
```

```
   xprev[0] = 5;
   xprev[1] = 6;
   xprev[2] = 10;

   for (int i=0; i<Nprev; i++){
       yprev[i]= f->Eval(xprev[i]);
       cout << "f(" << xprev[i] << ") = " << yprev[i] << endl;
   }
```

e i risultati sono riportati in Tab. 5.

n	$f(5)$	$f(6)$	$f(10)$
2	2.8177	2.5364	-1.0604
3	2.7518	2.2751	1.8881
4	2.9240	2.3097	5.1613

Tabella 5: Valori della funzione fittata.

Definita la distanza tra due punti della funzione come

$$d_{ab} = f(x_a, \hat{\boldsymbol{\theta}}) - f(x_b, \hat{\boldsymbol{\theta}}) \tag{12}$$

il Problema chiede di determinare la deviazione standard sulla distanza usando la propagazione dell'errore. Per fare ciò, ho usato la formula per la varianza di variabili correlate:

$$\sigma_d^2 = \sum_{i=0}^{n} \left(\frac{\partial d}{\partial \theta_i}\right)^2 \sigma_i^2 + \sum_{i \neq j}^{n} 2 \left(\frac{\partial d}{\partial \theta_i}\right)\left(\frac{\partial d}{\partial \theta_j}\right) \mathrm{cov}(\theta_i, \theta_j) = \sum_{i,j}^{n} \left(\frac{\partial d}{\partial \theta_i}\right)\left(\frac{\partial d}{\partial \theta_j}\right) \mathrm{cov}(\theta_i, \theta_j) \tag{13}$$

Questa formula di propagazione degli errori è una linearizzazione, ma nel caso in cui d sia una forma lineare nelle θ_i (come in questo caso), allora la formula è esatta. Ho implementato questo metodo riempiendo un vettore con le derivate di d rispetto ai θ_i

$$\frac{\partial d}{\partial \theta_i} = x_a^i - x_b^i \tag{14}$$

e calcolando i vari termini della 13 con un doppio ciclo `for`:

```
for (int i=0; i<npar ; i++){
        derivGth[i] = pow(x1,i)-pow(x2,i);
        cout << "dG/dtheta_"<<i << " = " << derivGth[i] << endl;
    }

    for (int i=0; i<npar; i++){
        for (int j=0; j<npar; j++){
            sigmaGsq += 2 * derivGth[i]*derivGth[j] * V[i][j];
        }
    }
    double sigmaG = pow (sigmaGsq,0.5);
```

I risultati per $n = 3$ sono mostrati in Tab. 6. Si osserva che più i valori (x_a, x_b) sono vicini, più la deviazione standard di d si abbassa.

x_a	x_b	$f(x_a)$	$f(x_b)$	d	σ_d
10	5	1.888	2.752	0.864	1.37
6	5	2.275	2.752	0.48	0.09
5	5	2.752	2.752	0	0

Tabella 6: Deviazione standard della distanza nel caso $n = 3$.

Infine, per il caso $n = 3$, ho definito una copia della funzione fissando i parametri ai valori

$$\boldsymbol{\theta}^T_{\mathrm{mod}} = (-0.75, 2.5, -0.5, 0.026) \tag{15}$$

Usando l'inverso della matrice di covarianza, ho calcolato il valore di χ^2 usando la formula

$$\chi^2 = \sum_{i,j=0}^{n} \left(\theta_{\text{mod},i} - \hat{\theta}_i\right) \left(V^{-1}\right)_{ij} \left(\theta_{\text{mod},j} - \hat{\theta}_j\right) \tag{16}$$

Ho implementato il calcolo di χ^2 e del corrispondente p-value con il codice

```
if (npar==4){

   thetaMod[0] = -0.75;
   thetaMod[1] = 2.5;
   thetaMod[2] = -0.5;
   thetaMod[3] = 0.026;

    for (int t=0; t<npar; t++) wf->SetParameter(t, thetaMod[t]);

    double chi2theta =0;

    for (int i=0; i<npar ; i++){
        for (int j=0; j<npar ; j++){

            chi2theta += (thetaMod[i] - thetaHat[i]) * Vinv[i][j] * (thetaMod[j]
    - thetaHat[j]);

        }
    }
    cout << "Chi2theta = " << chi2theta << endl;
    cout << "p-value = " << TMath::Prob(chi2theta, ndof);
}
```

ottenendo i valori numerici $\chi^2_\theta = 57.3$ e $p = 1.04 \cdot 10^{-11}$. Questi risultati, insieme con la Fig. 8, certificano come questo set di parametri non sia in buon accordo con il fit con i parametri ottimizzati.

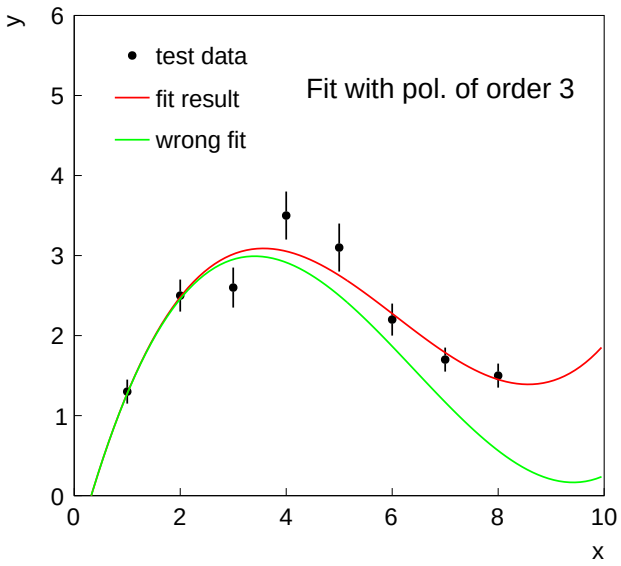


Figura 8: Fit di ML (rosso) e fit con parametri fissati (verde).

Problema 4

Esercizio 1

Questo esercizio consiste nell'eseguire un fit di Maximum Likelihood (ML) fittando un set di dati generati in modo casuale con una funzione del tipo

$$f(x;\theta,\xi) = \theta\frac{1}{\sqrt{2\pi\sigma^2}}\frac{1}{\xi}e^{-(x-\mu)^2/2\sigma^2} + (1-\theta)\frac{1}{\xi}e^{-x/\xi} \qquad 0 \leq x \leq x_{max} \tag{17}$$

Nell'esecuzione del programma ho prodotto i grafici della funzione fittata, della *Likelihood scan* per il parametro θ e del *contour plot* rappresentante l'equazione

$$-\log L = -\log L_{max} + \frac{1}{2} \tag{18}$$

nello spazio dei parametri (θ, ξ). Questo è l'unico spazio dei parametri disponibile in quanto tutti gli altri parametri sono fissati all'interno del programma. Questi grafici sono riportati in Fig. 9-10.

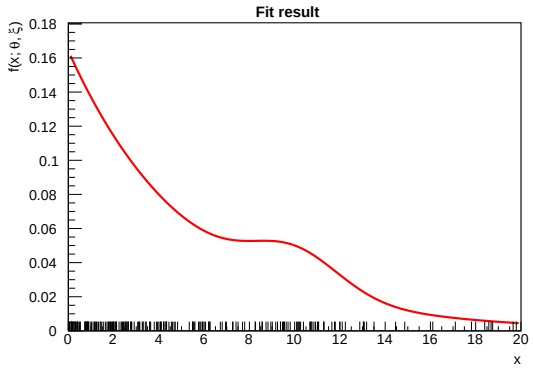
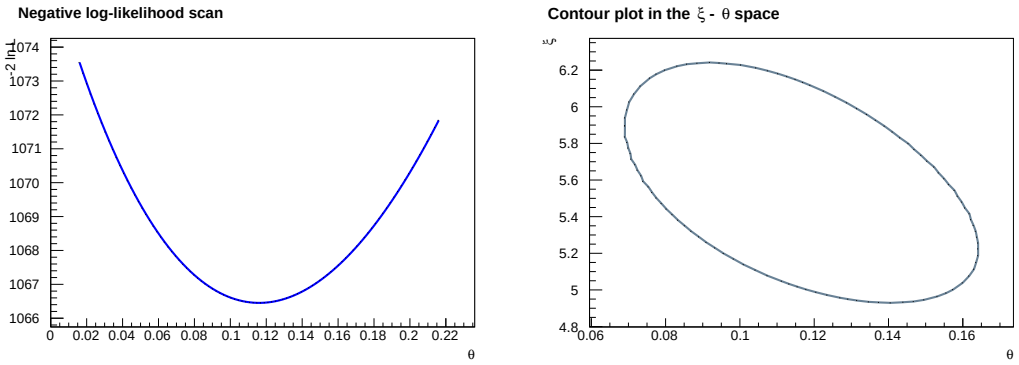


Figura 9: Risultati del fit di ML.

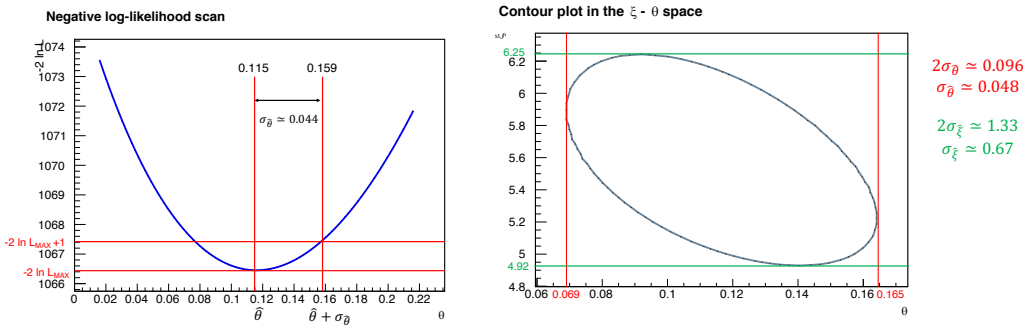


(a) Likelihood Scan

(b) Contour plot

Figura 10

Successivamente, viene chiesto di utilizzare un metodo grafico per ricavare le deviazioni standard sui parametri, e di confrontarlo con il metodo analitico. In particolare, dal grafico in figura 11a si può ricavare un valore di $\sigma_{\hat{\theta}}$ attraverso la 18. Dalla Fig. 11b tracciando le tangenti si possono ricavare stime di entrambe le deviazioni standard, stimate come metà della lunghezza dell'intervallo di confidenza. I risultati sono stampati in figura 11. La Tab. 7 riassume le stime ottenute con i diversi metodi.
Il punto b) del Problema chiede di mostrare che l'inversa della matrice di covarianza ha elemento di matrice proporzionale a n dimensione del campione, supponendo campione indipendente e



(a) Likelihood Scan

(b) Contour plot

Figura 11: Metodo grafico per estrarre σ.

Metodo	$\hat{\theta}$	$\sigma_{\hat{\theta}}$	ξ	$\sigma_{\hat{\xi}}$
Analitico	0.116	0.047	5.532	0.65
$\log L$ scan	0.115	0.044	-	-
Contour	0.117	0.048	5.58	0.67

Tabella 7: Stime dei parametri e delle loro incertezze.

identicamente distribuito. Questo è possibile ricordando

$$V_{ij}^{-1} = -E\left[\frac{\partial^2 \log L}{\partial\theta_i \partial\theta_j}\right] = -\int \frac{\partial \log L(\boldsymbol{x} \mid \boldsymbol{\theta})}{\partial\theta_i \partial\theta_j} L(\boldsymbol{x} \mid \boldsymbol{\theta}) d\boldsymbol{x} \tag{19}$$

con

$$L(\boldsymbol{x} \mid \boldsymbol{\theta}) = \prod_{i=1}^{n} f(x_i, \boldsymbol{\theta})$$

Utilizzando questa relazione otteniamo

$$V_{ij}^{-1} = \int \cdots \int -\frac{\partial^2}{\partial\theta_i \partial\theta_j} \left(\sum_{m=1}^{n} \log f(x_m; \boldsymbol{\theta})\right) \prod_{q=1}^{n} f(x_q, \boldsymbol{\theta}) dx_q =$$
$$= n \cdot \int -f(x; \boldsymbol{\theta}) \frac{\partial^2}{\partial\theta_i \partial\theta_j} \log f(x; \boldsymbol{\theta}) dx$$

Possiamo quindi supporre che $V_{ij} \propto n^{-1}$, ma siccome vale $V_{ij} \propto \sigma^2$ allora concludo che la deviazione standard di uno stimatore ML scala come $1/\sqrt{n}$.
Il punto c) del problema chiedeva di fare un plot della stima della deviazione standard del parametro θ in funzione del numero di valori generati, dunque presi in ingresso dal fit. Questo plot è mostrato in figura 12 e conferma le ipotesi ricavate prima, ovvero della dipendenza $1/\sqrt{n}$ della deviazione standard.
Infine, ho eseguito un fit variando il numero totale di parametri liberi. Questo è stato fatto modificando le righe di codice in cui venivano definiti i parametri e variando lo *step size* e il range. Mettendo entrambe queste quantità uguali a 0, il codice fissa il parametro ed esegue il fit.

```
1
2  par[0] = 0.4;
3    stepSize[0] = 0.04;
4    minVal[0] = 0.;
5    maxVal[0] = 1.;
6    parName[0] = "theta";
7
8    par[1] = 4.2;
9    stepSize[1] = 0.4;
10   minVal[1] =  0;
11   maxVal[1] =  10;
```

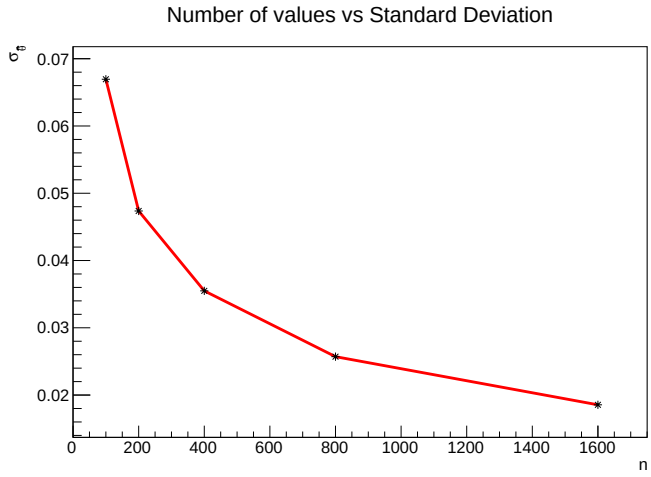


Figura 12: Calcolo di $\sigma_{\hat{\theta}}$ per diversi n

```
12  parName[1] = "xi";
13
14  par[2] = mu;
15  stepSize[2] = 0;      //  fixed
16  minVal[2] = 0.;
17  maxVal[2] = 0.;
18  parName[2] = "mu";
19
20  par[3] = sigma;
21  stepSize[3] = 0;      //  fixed
22  minVal[3] = 0.;
23  maxVal[3] = 0.;
24  parName[3] = "sigma";
```

Ho eseguito il codice con un numero crescente di parametri liberi, dal solo θ al set di 4 parametri liberi $(\theta, \xi, \mu, \sigma)$. I risultati sono riportati nel grafico 13. Non si notano significative variazioni di $\sigma_{\hat{\theta}}$ in funzione del numero di parametri liberi, anche se si può osservare una leggero trend decrescente (all'ordine della terza cifra significativa) all'aumentare del numero di parametri.

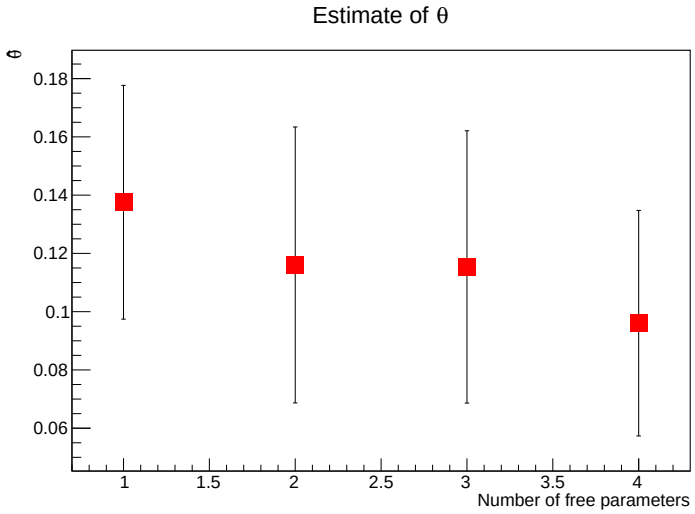


Figura 13: $(\hat{\theta}, \sigma_{\hat{\theta}})$ al variare del numero di parametri liberi.

Problema 5

Nota: Ho prodotto interamente il codice per svolgere i Problemi 5 e 6. Per ragioni di comodità, ho scelto di non allegarlo alla relazione. Il codice è disponibile qui.

Esercizio 1

Questo esercizio riguarda il test di ipotesi e il Look Elsewhere Effect. A lezione è stato presentato uno studio con un fit di tipo binned di un set di dati. All'interno di questi dati, un bin contiene un numero molto diverso di eventi rispetto al resto dell'istogramma. Viene chiesto di valutare se questa disparità è dovuta ad una fluttuazione statistica oppure se sono presenti eventi di segnale all'interno del campione.
Nel codice che ho implementato, la null hypothesis H_0 è l'ipotesi di solo segnale; possiamo quindi considerare gli eventi distribuiti in modo uniforme. L'ipotesi alternativa H_1 include la presenza di un segnale gaussiano. Nel codice ho costruito due modelli, uno per il solo fondo e uno che include il segnale. Per fare questo ho creato due pdf opportune. Le due pdf sovrapposte sono mostrate in Fig. 14.

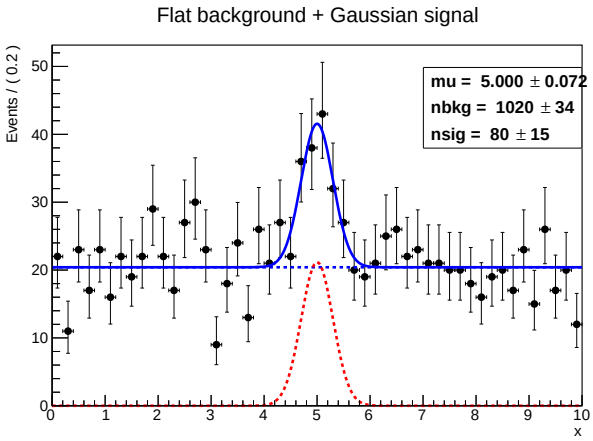


Figura 14: Sovrapposizione dei due modelli.

Successivamente, ho calcolato la significanza globale in modo naive, e anche tenendo conto del Look Elsewhere Effect. Per fare questo ho fatto uso della classe di `RooStats HypoTestCalculator`. I risultati sono mostrati in Fig. 15.

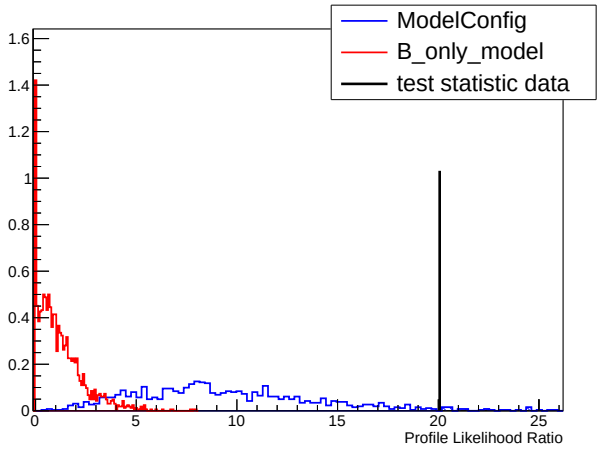


Figura 15: Sovrapposizione dei due modelli.

Esercizio 2

Il secondo esercizio chiede di utilizzare il dataset dell'esercizio precedente per calcolare gli intervalli di confidenza su due parametri: il numero di eventi di segnale e la media del picco gaussiano. Per fare questo ho fatto uso della libreria `RooStats` e ho implementato un codice che restituisce gli intervalli di confidenza. In particolare, ho definito un `RooWorkspace` per importare il dataset e i parametri di interesse; successivamente ho fatto uso della classe `ProfileLikelihoodCalculator` per ottenere gli intervalli di confidenza ad un CL desiderato. I risultati ottenuti sono mostrati in Tabella 8.

CL	n_{sig}	μ
68%	[65.2136, 94.521]	[4.92778, 5.07214]
90%	[56.3636, 104.551]	[4.87951, 5.11929]
95%	[52.1366, 109.566]	[4.85513, 5.14287]

Tabella 8: Intervalli di confidenza.

Ho anche prodotto un plot del contour dei tre livelli di confidenza nel piano (n_{sig}, μ), che è mostrato in figure 16. Le curve sono di forma abbastanza circolare. Inoltre, le curve non sono orientate lungo gli assi $y = \pm x$. Dunque, possiamo approssimare le due variabili come scorrelate.

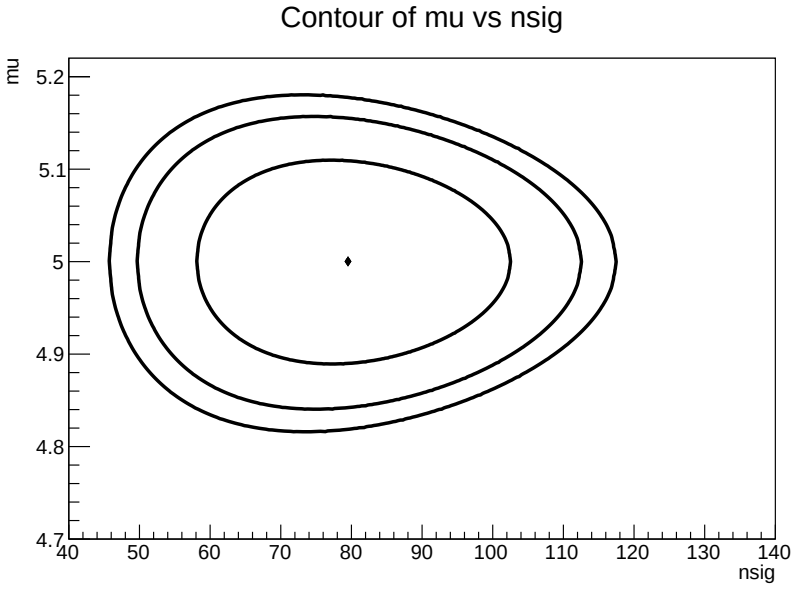


Figura 16: Contour plot per i tre livelli di confidenza.

Problema 6

Questo esercizio riguarda gli intervalli di confidenza in un esperimento in cui sono presenti eventi di segnale e di background. L'obiettivo è quello di determinare l'intervallo di confidenza per il numero di eventi di segnale nel caso in cui il numero di eventi di background e il suo errore sono conosciuti (distribuiti in modo gaussiano).

Parte 1: Approccio bayesiano

L'approccio bayesiano alla determinazione degli intervalli di confidenza prevede la conoscenza della *posterior pdf*. Se questa è conosciuta, allora gli intervalli di confidenza possono essere determinati con facilità. La formula per ricavare la posterior pdf è, assumendo che la *prior pdf* $\pi(s)$ per il segnale sia uniforme:

$$p(s) = \frac{L(s,b)\pi(b)}{\iint L(s,b)\pi(b)dsdb} \tag{20}$$

dove la likelihood L è data dal prodotto tra una Poissoniana con valore atteso $s+b$ e la prior pdf, ovvero il vincolo di gaussianità su b, con errore σ_b. Una volta che la posterior pdf $p(\theta \mid x)$ è conosciuta, si possono calcolare l'intervallo di confidenza o l'upper limit per il numero di eventi di segnale, attraverso le relazioni

$$\alpha = \int_{-\infty}^{a} p(\theta \mid x)d\theta \qquad \beta = \int_{b}^{+\infty} p(\theta \mid x)d\theta \tag{21}$$

Allo stesso modo, l'upper limit s^{up} a un livello di confidenza $1-\beta$ si calcola risolvendo (numericamente) l'equazione

$$1-\beta = \frac{\int_0^{s^{\text{up}}} L(n \mid s)ds}{\int_0^{\infty} L(n \mid s)ds} \tag{22}$$

Inoltre, la posterior pdf per il numero di eventi di segnale dipende anche dall'errore sul numero di eventi di background. Infatti, se l'errore sul parametro b è minore, qualitativamente si hanno più informazioni su s. Questa intuizione è confermata dal grafico della posterior pdf per diversi valori di σ_b. Per errori bassi, la funzione è più piccata, e di conseguenza l'upper limit si avvicina al valore vero (vedi Eq. 22).

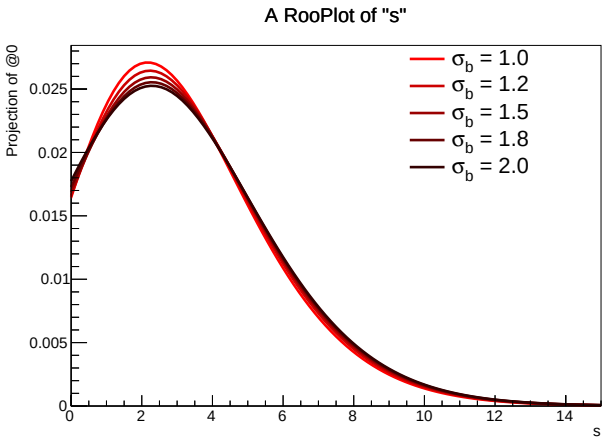


Figura 17: Posterior pdf di s per diversi σ_b.

Parte 2: Approccio frequentista

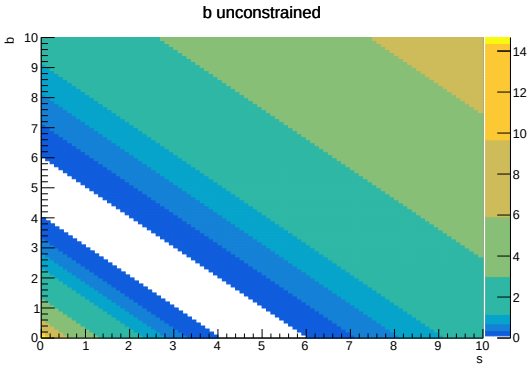
Se si affronta lo stesso problema con un approccio frequentista, il primo passo è come nel caso precedente calcolare la likelihood, data dalla distribuzione poissoniana

$$L(s,b) = P(n \mid s,b) = \frac{(s+b)^n}{n!} e^{-(s+b)} \tag{23}$$

e si definisce il test statistico con il *likelihood ratio*:

$$t = -2\frac{L(s,b}{L_{\max}} \tag{24}$$

Si cerca di minimizzare t in quanto questo significa far avvicinare la likelihood al suo valore massimo. Generalmente si può usare il Teorema di Wilks per approssimare t come una variabile χ^2 e di conseguenza estrarre un livello di significanza per diversi valori di t, ma in questo caso il numero atteso di eventi di segnale è vicino allo 0 (vedi Parte 1) e dunque il Teorema non è utilizzabile.
Poiché il numero totale di eventi $n = s + b = 5$ è conosciuto, si può fare uno scan di t per diversi valori di s e b e osservare in quale zona possiamo aspettarci gli intervalli di confidenza:



Per ovviare al problema di campione ridotto, si generano una serie di pseudo esperimenti assegnando ad ogni esperimento una coppia di valori (s,b). Si intende che questo metodo è computazionalmente molto pesante, specialmente se comparato con l'approccio bayesiano.
Come nel caso precedente, possiamo acquisire informazione su s mettendo dei vincoli su b, come ad esempio un vincolo gaussiano (proprio come nell'approccio bayesiano). In questo caso la likelihood è dato dal prodotto tra la poissoniana già nota e una gaussiana per il fondo. In figura 18 è plottata la quantità $t/2$ per diversi valori di σ_b. Si osserva che l'area con il test statistico più basso è più ristretta quando l'errore è più basso. Un constraint su b incide quindi positivamente sulla misura del parametro di interesse.
Un altro metodo per verificare questo è quello della *Profiled Likelihood*, ovvero fare un plot monodimensionale di $t(s) = -2\log\frac{L(s,\hat{\hat{b}})}{L_{\max}}$, ovvero si fissa un valore di s e si ottiene $\hat{\hat{b}}$ dal fit di ML. Dalla figura 19 si osserva come a maggiori σ_b corrispondano maggiori upper limit.

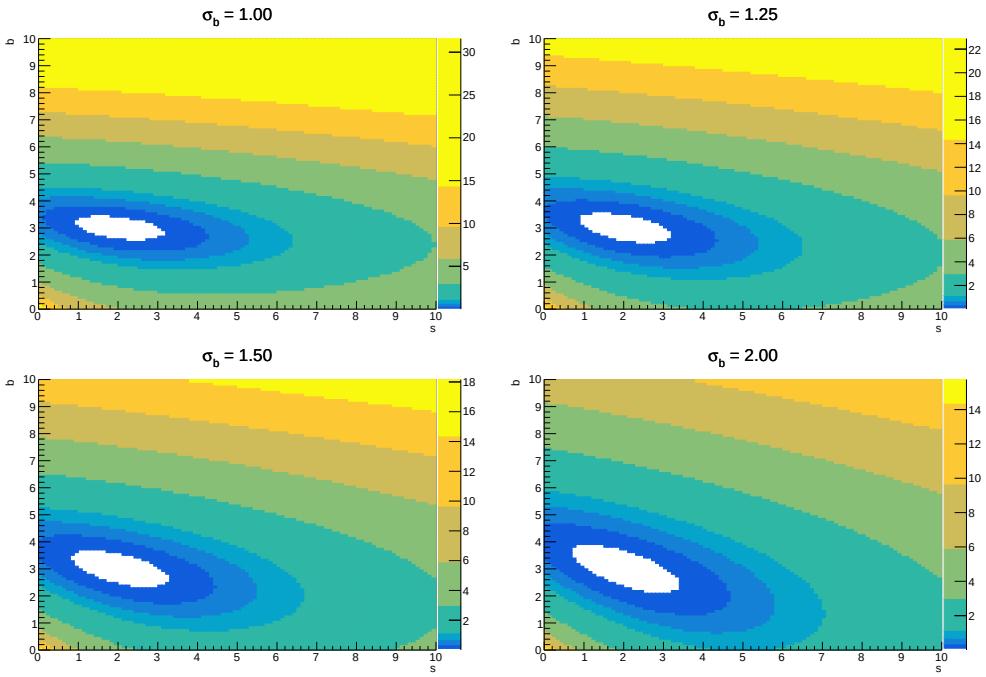


Figura 18: 2D plot di t per diversi σ_b.

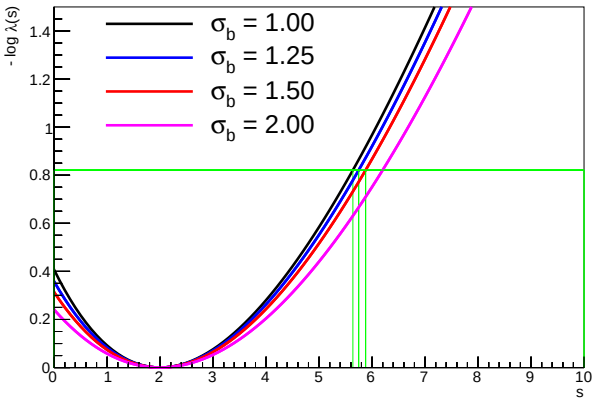


Figura 19: Upper limit per diversi σ_b.