Calcolo applicato alla Statistica
Maximum Likelihood

Problema fisico – 1/2

- Consideriamo un esperimento consistente nella misura, per un tempo **T** fissato, delle trasmutazioni nucleari (spontanee o indotte) di un nuclide in un altro nuclide.
  - Chiamiamo **n** il numero di trasmutazioni osservato, ossia **n** è il numero di conteggi osservato nel tempo **T** con un apposito rivelatore.
- Ora ripetiamo la misura per un numero **I** di volte, sempre nelle stesse condizioni.
  - Al termine dell'esperimento avremo una sequenza di valori **n**, ed a ciascun valore è associata una frequenza k_n.
  - Avremo quindi che $\sum_{n=0}^{N} k_n = I$, e **N** è il massimo dell'intervallo esplorato per **n** (in teoria infinito)

Problema fisico – 2/2

- Tabella dei risultati:

Conteggi n	0	1	2	3	4	5	6	7	8	9
Frequenza k_n	1	3	4	5	3	2	1	1	0	0

- Dato che il numero di conteggi (successi) è piccolo rispetto al numero di tentativi (come per esempio nel caso di una reazione nucleare), si suppone che il campione ottenuto segua la **statistica di Poisson**.

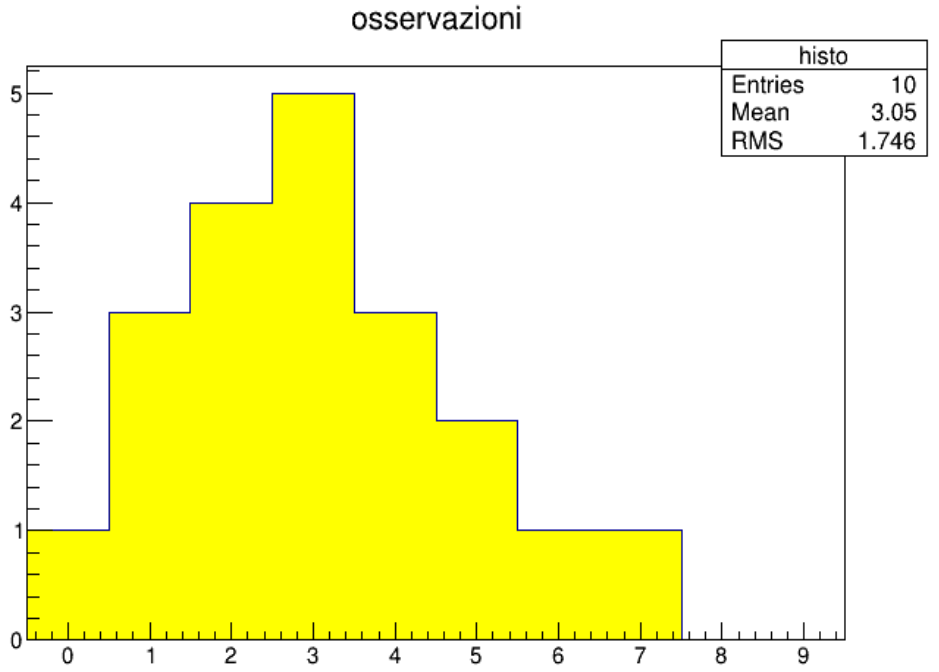
Esercizio 1

- **Rappresentare** graficamente **l'istogramma dei risultati** usando ROOT.

```
int O[] = { 1, 3, 4, 5, 3, 2, 1, 1, 0, 0 };
int s = sizeof(O) / sizeof(O[0]);
int N = s - 1;

// Disegno dell'istogramma
TH1F *histo = new TH1F("histo", "osservazioni", s, -0.5, s-0.5);
for (int i = 0; i < s; ++i) {
        histo->Fill(i, O[i]);
}

histo->SetFillColor(5);
histo->Draw();
```



Esercizio 2

- **Calcolare** con le note formule di statistica la nostra miglior stima del **valor medio** μ e della **varianza** σ^2.

```
int sommatoria(int array[], int size) {
        if (size <= 0) return -1;
        int cursum = 0;
        for (int i = 0; i < size; ++i) {
                cursum += array[i];
        }
        return cursum;
}
```

```
int I = sommatoria(O, s);

// Calcolo di media e deviazione standard
double m = 0;
for (int i = 0; i < s; ++i) {
        m += i * O[i];
}
m /= I;

double mediaScarti = 0;
for (int i = 0; i < s; ++i) {
        mediaScarti += O[i] * pow(i - m, 2);
}
mediaScarti /= (I-1);
double sigma = sqrt(mediaScarti);
```

$$M_a = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\sigma_x = \sqrt{\frac{\sum_{i=1}^{N}(x_i - M_a)^2}{N}}$$

Esercizio 3 – 1/3

- Ottenere il **valore di μ** con il metodo della massima verosimiglianza (**Maximum Likelihood**), utilizzando un algoritmo per la ricerca di un estremo di una funzione in 1 dimensione; la funzione è:

$$y = \log\left(L\left(k_n;\mu\right)\right) \quad ; \quad \mu = [0,\ 9] \quad \text{con} \quad L\left(k_n;\mu\right) = \prod_{n=0}^{N}\left(\frac{e^{-\mu}\mu^{n}}{n!}\right)^{k_n}$$

  - confrontare il risultato ottenuto con la media calcolata al punto 1).

Esercizio 3 – 2/3

- Iniziamo con l'implementare le funzioni accessorie necessarie:
  - funzione **fattoriale**
  - funzione **produttoria**
  - funzione **likelihood**

$$y = \log\left(L(k_n;\mu)\right) \quad ; \quad \mu = [0, 9]$$

$$L(k_n;\mu) = \prod_{n=0}^{N}\left(\frac{e^{-\mu}\mu^n}{n!}\right)^{k_n}$$

```
int fact(int n) {
        if (n <= 1) return 1;
        return fact(n-1) * n;
}

double likelihood(double mu, int n, int freq) {
        double val = exp(-mu) * pow(mu, n);
        val /= fact(n);
        val = pow(val, freq);
        return val;
}

double func_likelihood(double mu) {
        double l = 1;
        for (int n = 0; n < s; ++n) {
                l *= likelihood(mu, n, O[n]);
        }
        return log(l);
}
```

Esercizio 3 – 3/3

- Ora usiamo le tecniche algoritmiche per la **ricerca del massimo**.
  - Possiamo usare il **metodo della trisezione**!!

```
double max_trisezione(double x0, double x1, double precision) {
        if ((fabs(x1 - x0)) < precision) return 0.5 * (x0 + x1);
        double x2 = x0 + ((x1 - x0) / 3.);
        double x3 = x1 - ((x1 - x0) / 3.);
        if (func_likelihood(x2) < func_likelihood(x3)) {
                return max_trisezione(x2, x1, precision);
        }
        else {
                return max_trisezione(x0, x3, precision);
        }
}
```

```
// media calcolata con maximum likelihood
double ml = max_trisezione(0, N, 0.000001);
```

Esercizio 4 – 1/2

- detto μˆ il valore ottenuto al punto 3, ottenere i valori $\mu+$ e $\mu-$ per cui:

$$\log\left(L\left(k_n;\mu_+\right)\right) = \log\left(L\left(k_n;\hat{\mu}\right)\right) - \frac{1}{2} \quad ; \quad \log\left(L\left(k_n;\mu_-\right)\right) = \log\left(L\left(k_n;\hat{\mu}\right)\right) - \frac{1}{2}$$

  - Questi valori ci consentono di calcolare gli **errori sinistro e destro** in base a quanto previsto dalla teoria.

$$\sigma_- = \mu_- - \hat{\mu} \quad ; \quad \sigma_+ = \mu_+ - \hat{\mu}$$

  - Si utilizzi un **algoritmo per la ricerca degli zeri di una funzione** in 1 dimensione; la funzione è:

$$y = \log\left(L\left(k_n;\mu\right)\right) - \left(\log\left(L\left(k_n;\hat{\mu}\right)\right) - \frac{1}{2}\right)$$

  e le due regioni di ricerca sono: $\mu < \hat{\mu}$; $\mu > \hat{\mu}$

Esercizio 4 – 2/2

- Usiamo il metodo della **bisezione ricorsiva** e creiamo un metodo accessorio per calcolare gli errori:

```
double zero_R_bisezione(const double muhat, double xMin, double xMax, double precision) {
        double x_m = 0.5 * (xMin + xMax);
        if ((xMax - xMin) < precision) return x_m;
        if (errori(x_m, muhat) * errori(xMin, muhat) < 0) {
                return zero_R_bisezione(muhat, xMin, x_m, precision);
        }
        return zero_R_bisezione(muhat, x_m, xMax, precision);
}

double errori(double mu, const double muhat) {
        double val = func_likelihood(mu);
        val -= func_likelihood(muhat);
        val += 0.5;
        return val;
}
```

- Calcoliamo quindi **errore destro** e **errore sinistro**:

```
// calcolo errore destro e errore sinistro
double mum = zero_R_bisezione(ml, 0, ml, 0.000001);
double mup = zero_R_bisezione(ml, ml, N, 0.000001);
double es = ml - mum;
double ed = mup - ml;
```

Esercizio 5 – 1/2

- **Calcolare il χ^2** del risultato ottenuto, nelle due ipotesi:
  - si ignorino i bin in cui la frequenza è 0;
  - si assuma come errore per n=0: σ = 1 .

- Negli altri casi si usi per ciascun bin l'errore $\sigma = \sqrt{k_n}$ che discende dalla varianza per la distribuzione di Poisson (anche la frequenza in ciascun bin fluttua in modo Poissoniano, $\sigma^2 = k_n$).

Esercizio 5 – 2/2

- χ^2 calcolato nelle due ipotesi:

```
// calcolo del chi quadro, ignorando i bin con frequenza 0
double chi1 = 0;
for (int n = 0; n < s; ++n) {
        if (O[n] != 0) {
                double expected = likelihood(ml, n, 1);
                double val = O[n] - I*expected;
                val = pow(val, 2);
                val /= O[n];
                chi1 += val;
        }
}

// calcolo del chi quadro, con errore 1 per i bin con frequenza 0
double chi2 = 0;
for (int n = 0; n < s; ++n) {
        double expected = likelihood(ml, n, 1);
        double val = O[n] - I*expected;
        val = pow(val, 2);
        if(O[n] != 0) val /= O[n];
        if(O[n] == 0) val /= 1;
        chi2 += val;
}
```

Esercizio 6 – 1/2

- calcolare la **probabilità** che una ripetizione dell'esperimento dia un **risultato** $\mu\hat{}$ compreso tra $\mu-$ e $\mu+$ (**"intervallo di confidenza"**), integrando la funzione:

$$L(k_n;\mu) = \prod_{n=0}^{N}\left(\frac{e^{-\mu}\mu^n}{n!}\right)^{k_n} = P(\mu \mid k_n)$$

sull'intervallo e normalizzando opportunamente

Esercizio 6 – 2/2

- Per calcolare l'**integrale** usiamo il **metodo dei rettangoli!**

```
double integrale_rettangoli(double xMin, double xMax, int N_bin) {
        double step = (xMax - xMin) / double(N_bin);
        double integrale = 0;
        double x = xMin;
        while (x < xMax) {
                double h = exp(func_likelihood(0.5 * (x + x + step)));
                integrale += h * step;
                x += step;
        }
        return integrale;
}
```

- Il risultato va quindi **normalizzato** rispetto all'**area dell'intera curva**, infatti la funzione di likelihood come funzione di μ non è una p.d.f, quindi in generale $\int_{-\infty}^{+\infty} d\mu\, L(k_n, \mu) \neq 1$

```
// Probabilita' che l'esperimento dia un risultato compreso tra mu- e mu+
double p = integrale_rettangoli(mum, mup, 1000);
p /= integrale_rettangoli(0, s, 1000);
```

Esercizio 7 – 1/2

- Effettuare il **fit dei dati** nelle due ipotesi:
  - che la distribuzione dei risultati sia **Gaussiana**
  - che la distribuzione dei risultati sia **Poissoniana**

- Confrontare i risultati dei due fit.

Esercizio 7 – 2/2

- **Fit Gaussiano**:

```
// Fit con gaussiana
TCanvas *c1 = new TCanvas();
TF1* gauss = new TF1("gauss", "gaus", 0, s);
gauss->SetParameter(1, m);
gauss->SetParameter(2, m);
histo->Fit("gauss", "L");
histo->SetFillColor(4);
histo->Draw();
```

- **Fit Poissoniano**:

```
double poissonf(double *x, double *par) {
        return par[0]*TMath::Poisson(x[0],par[1]);
}
// Fit con poissoniana
TCanvas *c2 = new TCanvas();
TF1* pois = new TF1("pois", poissonf, 0, s, 2);
pois->SetParameter(0, I);
pois->SetParameter(1, m);
histo1->Fit("pois", "L");
histo1->SetFillColor(5);
histo1->Draw();
```

Esercizio 8 – 1/2

- Confrontare il **fit "manuale"** effettuato con il metodo della Massimo Likelihood con il fitting **Poissoniano**.
  - Per far questo usiamo la funzione $P(x) = \frac{e^{\hat{\mu}}\,\hat{\mu}^{x}}{\Gamma(x+1)}$

    che è la distribuzione di Poisson generalizzata a variabile continua (*x* al posto di *n*).
  - Una volta definita la funzione la possiamo disegnare utilizzando l'oggetto **TGraph** di ROOT.
  - Specificando il parametro **"Same"** al metodo **Draw** possiamo effettuare il disegno sullo stesso canvas su cui è disegnata la funzione Poissoniana fittata da ROOT per confrontare visivamente i due risultati.

Esercizio 8 – 2/2

- Ecco il risultato:

```
double *xg = new double[s*10];
double *yg = new double[s*10];
for (int i = 0; i < s*10; ++i) {
        xg[i] = i/10.;
        yg[i] = I*exp(-ml)*pow(ml, xg[i])/TMath::Gamma(xg[i]+1);
}
TGraph *gr1 = new TGraph(s*10, xg, yg);
gr1->SetLineColor(1);
gr1->SetLineWidth(2);
gr1->Draw("same");
```