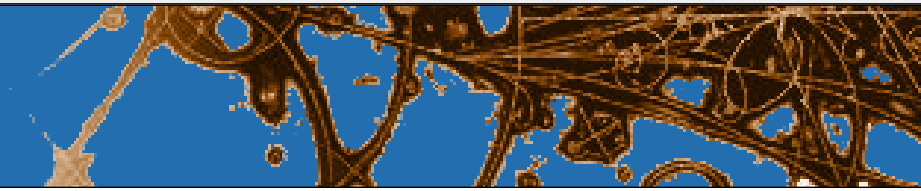


# Introduzione all'Analisi Multivariata

## I. Analisi a tagli

Pietro Biassoni[1,2]

[1]INFN Milano

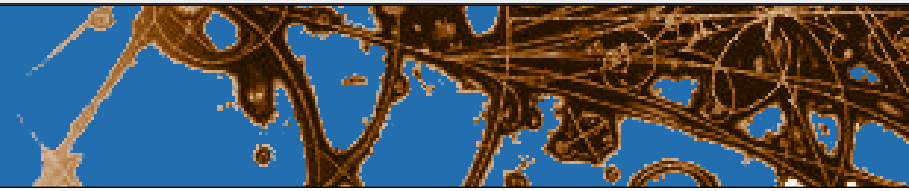
[2]Dipartimento di Fisica
Università degli Studi di Milano

Metodologie di Analisi dei Dati
18 Dicembre 2009

# Outline

# Outline

# Analisi a Tagli



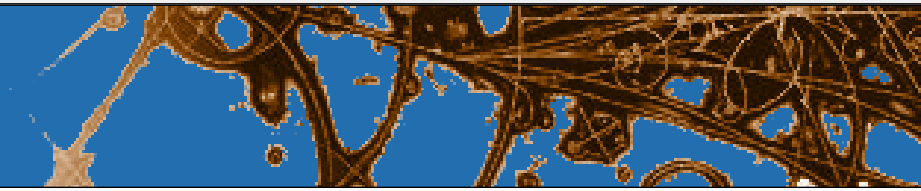
Lo scopo della lezione di oggi è di effettuare una analisi a tagli.

Il campione che ci apprestiamo ad utilizzare
`/home/comune/AnalisiDati/mva1_sample.root`
contiene due variabili che hanno un buon potere discriminante tra segnale e fondo.

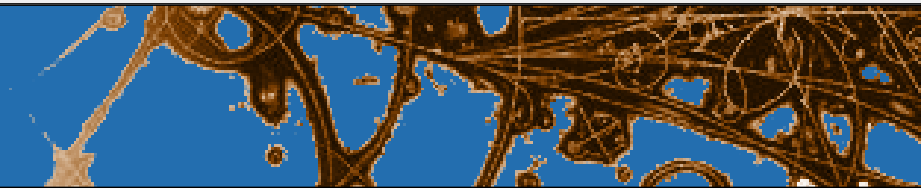
**Cosa impareremo a fare:**

- Presentare le variabili in modo comprensibile.
- Ottimizzare un taglio in una dimensione.
- Ottimizzare un taglio in due dimensioni.

# Outline



# Introduzione

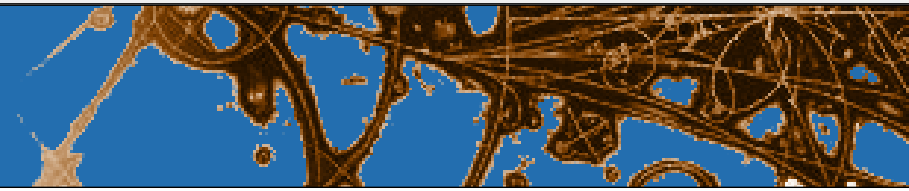


Nel nostro rile `.root` esistono due alberi: *signal* e *bkg*. Ciascun albero contiene due variabili.

**Cosa dobbiamo fare:**

- Aprire i files e accedere agli alberi
- Plottare le variabili in modo da avere un confronto grafico.

# Accede agli alberi



Dobbiamo aprire il file root e recuperare gli alberi

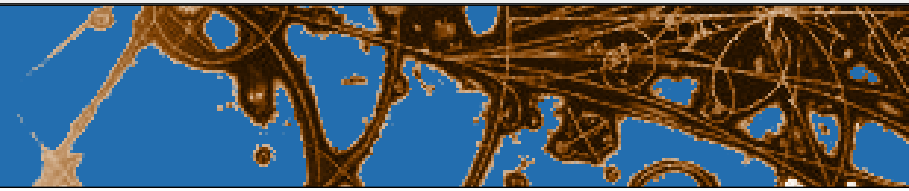
```
TFile *f = new TFile("mva1_sample.root","READ");
TTree *tsig = (TTree*)f->Get("signal");
TTree *tbkg = (TTree*)f->Get("bkg");
```

a questo punto vogliamo disegnare le variabili. La cosa piu' semplice da fare è usare il metodo di `TTree`

```
Draw(const char* varexp, const TCut& selection, Option_t*
option = "");
```

varexp è il nome della variabile che volete disegnare (es. `"var1"`);
selection è una espressione che viene interpretata come un taglio (es. `"var2<0.5"`): verranno disegnati solo i dati che soddisfano questo taglio;
option sono le opzioni per la grafica (ad es. `"SAME"` per disegnare immagini sovrapposte).

# TCanvas



La TCanvas è l'oggetto che ROOT usa per contenere disegni

```
TCanvas *c1 = new TCanvas("name","title",1);
```

un metodo utile della TCanvas è

```
c1->Divide(ncolonne,nrighe);
```

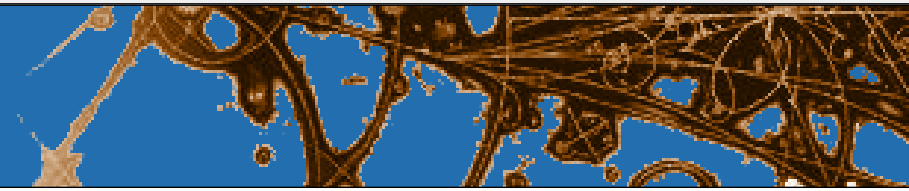
in questo modo la TCanvas è divisa in *pads*, che sono numerati da 1 a $n$, da sinistra a destra dall'alto in basso.
Per disegnare due oggetti in pads diversi

```
c1->cd(1); //scelgo il pad 1
tree->Draw("var1"); //disegno var1 nel pad 1
c1->cd(2); //scelgo il pad 2
tree->Draw("var2"); //disegno nel pad 2
```

# Esercizio

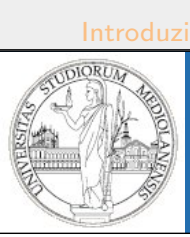
Aprire il file mva1_sample.root, recuperare gli alberi e disegnare var1 e var2, nei due pad di una canvas. In ogni pad disegnare sia la distribuzione del segnale che quella del fondo

# Esercizio

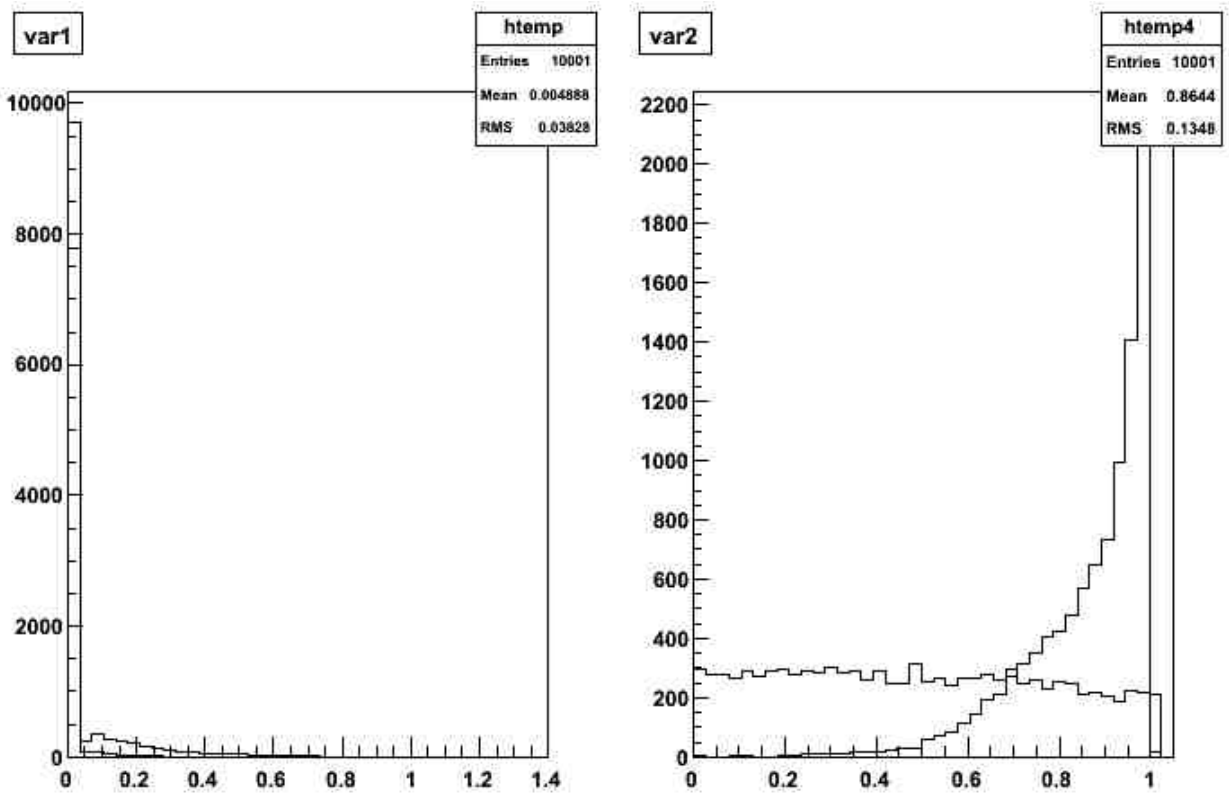


```
TFile *f = new TFile("mva1_sample.root");
TTree *tsig = (TTree*)f->Get("signal");
...
TCanvas *c1 = new TCanvas(...);
c1->Divide(2);

c1->cd(1);
tsig->Draw("var1");
tbkg->Draw("var1","","SAME");
...
```



# Esercizio




Questo plot è decisamente bruttino. Non si capisce che variabili sono plottate, non si distinguono le due distribuzioni.

# Recuperare gli istogrammi

Se vogliamo utilizzare le opzioni grafiche degli isogrammi, dobbiamo avere accesso ad essi:

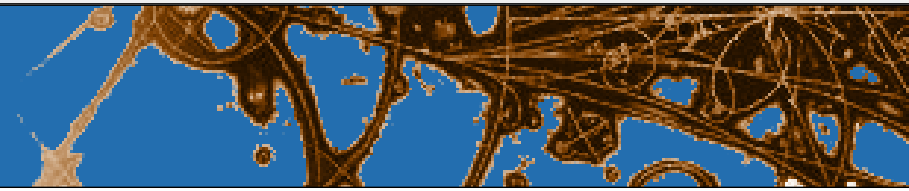
```
tsig->Draw("var1>>htemp(40)");
```

fa si che l'istogramma sia disegnato in un istogramma di nome "htemp" di 40 bin.
Per recuperarlo:

```
TH1F* htemp = (TH1F*)gROOT->FindObject("htemp");
htemp-> ...  //Operazioni sull'istogramma
```

Ovviamente se dobbiamo disegnare 4 istogrammi dobbiamo usare 4 nomi diversi e recuperarli tutti e 4 con 4 puntatori TH1F* diversi...

# Opzioni grafiche



Togliere il colore grigio alla canvas

```
gROOT->SetStyle("Plain");
```

Cambiare colore/stile alla linea (Attenzione: è necessario cambiare non solo il colore, ma anche lo stile nel caso in cui chi legge stampi in b/n...)

```
htemp->SetLineColor(N); //N è un intero
htemp->SetLineStyle(N); //N è un intero
```

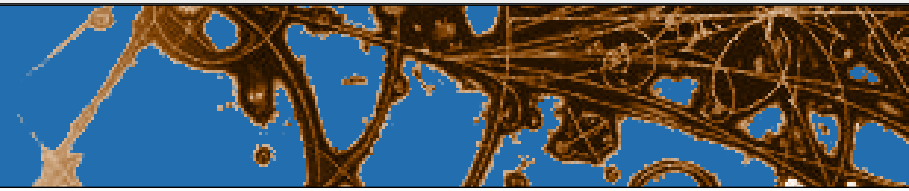
Togliere le statistiche

```
htemp->SetStats(kFALSE);
```

Cambiare il range dell'asse, assegnare un titolo

```
htemp->GetXaxis()->SetRangeUser(min,max);
htemp->GetXaxis()->SetTitle("pippo");
```

# Legenda



Una cosa utilissima è aggiungere una legenda al grafico

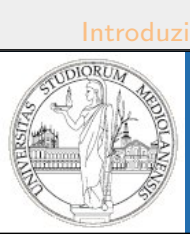
```
TLegend *leg =new TLegend(0.1,0.1,0.2,0.2);
```

Per aggiungere voci alla legenda

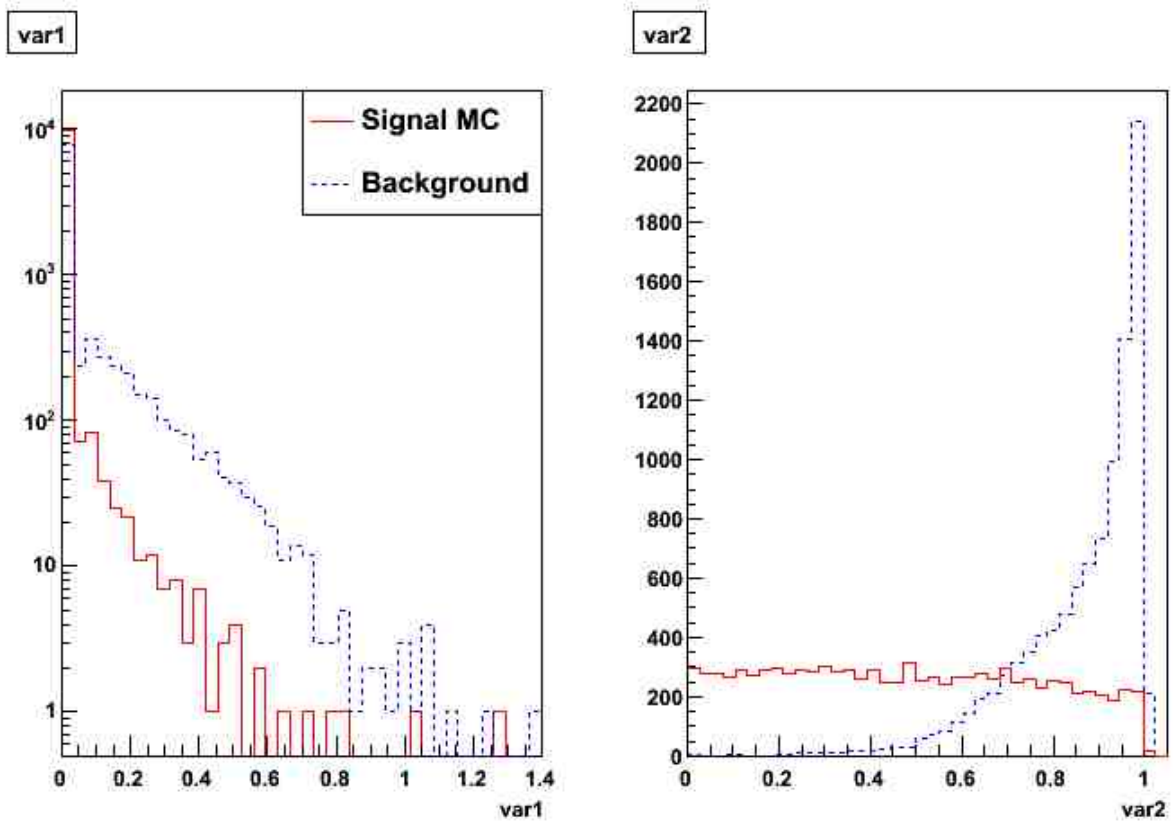
```
leg->AddEntry(htemp,"Segnale","L")
```

L'ultimo argomento indica il tipo di simbolo nella legenda: "L" = linea, "P" = punto, "F" = riempimento.

```
c->cd(1) // Seleziono il pad
leg->Draw();
leg->SetFillColor(0); // Colore dello sfondo
leg->SetTextSize(0.05); // Grandezza carattere
```



# Presentazione delle variabili

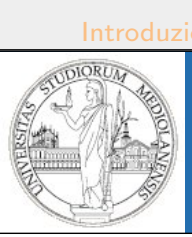



Con un po' di lavoro questo è il risultato

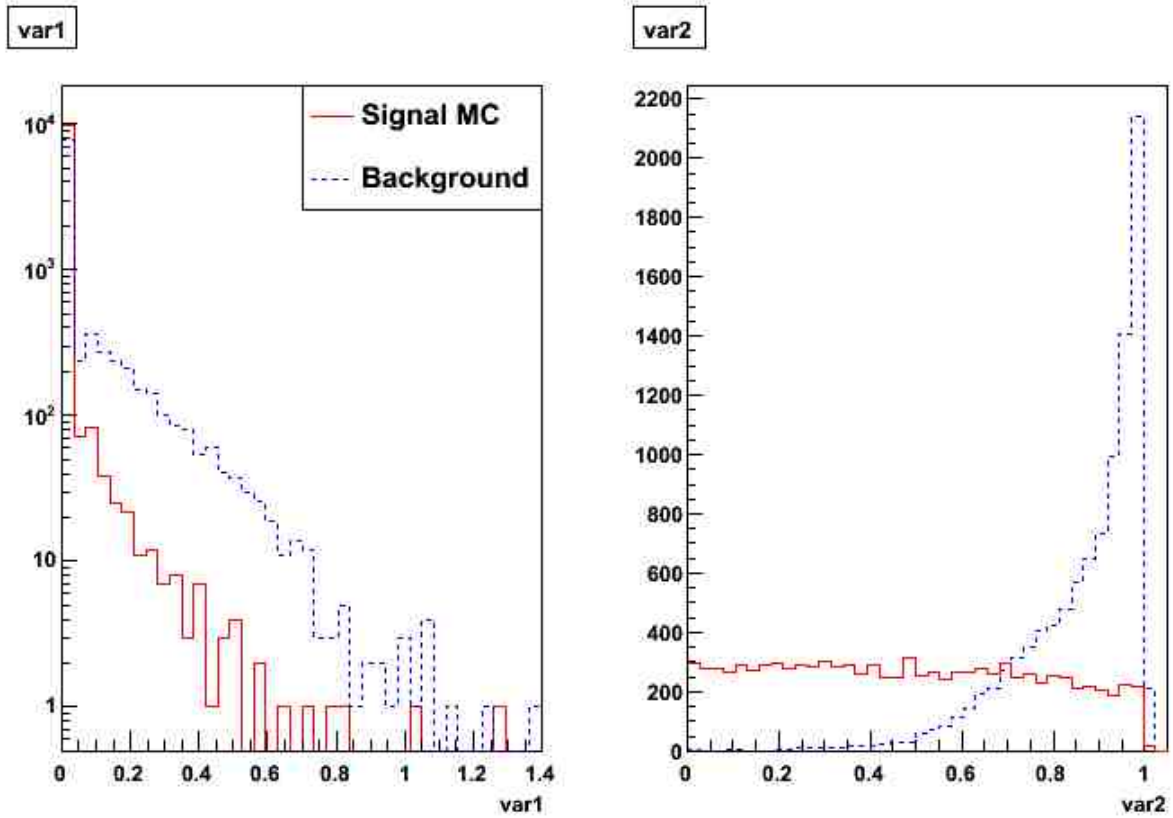
# Esercizio (a casa)

Rendete il vostro plot più presentabile

# Outline



# Introduzione




Vogliamo ora ottimizzare un taglio su $var_1$ e $var_2$ *separatamente*.
Volgiamo cioè trovare un taglio che ottimizzi una certa figura di merito (FOM) per ciascun taglio

# Ottimizzazione della Significanza

Vogliamo ottimizzare la figura di merito

$$FOM = \frac{n_S}{\sqrt{n_S + n_B}}$$

dove $n_S$ ($n_B$) è il numero di eventi che passa il taglio.
Come traduciamo questo in codice?
Vogliamo

- fare un *ciclo* e provare diversi valori del taglio
- per ogni taglio ottenere $n_S$ e $n_B$
- per ogni taglio calcolare $\epsilon_{S,B} = n_{(S,B)}/N^{tot}_{(S,B)}$

Per ottenere il numero di eventi che superano il taglio basta fare

```
int s = tsig->Draw("var1","var1<0.2");
```

# Ottimizzazione della Significanza

## String

```
int s = tsig->Draw("var1","var1<0.2");
```

il valore del taglio deve cambiare *ad ogni ciclo*.

Si puo' usare la funzione *sprintf* (ma è C !) In C++ si usano string e stringstream.

`string` è un contenitore di carattere, ma ha il difetto di non potervi inserire variabili. Cioè non è possibile fare

```
double vcut = 0.1; //valore del taglio da incrementare
string cut; //dichiaro la stringa
cut="var1<"; //inserisco "var1<" nella stringa
cut+=vcut; //NON FUNZIONA!!!
```

# Ottimizzazione della Significanza

## Stringstream

`stringstream` è un contenitore usato per riempire le stringhe con valori variabili.
Ha la stessa sintassi di `cout`.

```
double vcut = 0.1; //valore del taglio
...
for(int i ......){ //ciclo
stringstream buffer; //dichiaro stringstream
buffer <<"var1<" <<vcut;
int s = tsig->Draw("var1",(buffer.str()).c_str());
...
vcut+=....  //incremento del taglio
...
} //fine del ciclo
```

`buffer.str()` trasforma `stringstream` in `string`
`.c_str()` trasforma `string` in `char*`

# Ottimizzazione della Significanza

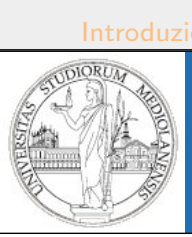
## Plot

Per ciascuna delle variabili vogliamo avere un plot di

- Efficienza del segnale vs valore del taglio
- Reiezione $(1 - \epsilon)$ del fondo vs valore del taglio
- FOM vs valore del taglio
- ROC curve: reiezione del fondo vs efficienza del segnale

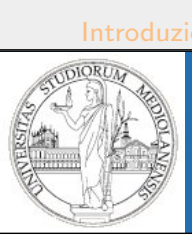
Quindi a ogni step del ciclo vogliamo riempire 4 `TGraph`

```
for(int i...){ //ciclo
...
gr1->SetPoint(i,vcut,eff_signal); //grafico eff.  segnale
gr2->SetPoint(i,vcut,rej_bkg); //reiezione bkg
gr3->SetPoint(i,vcut,FOM); //grafico FOM
gr4->SetPoint(i,eff_signal,rej_bkg); //ROC curve
...
} //fine del ciclo
```

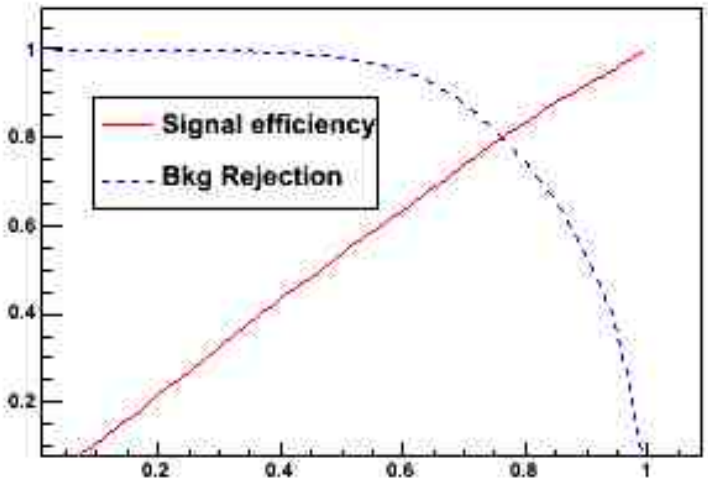


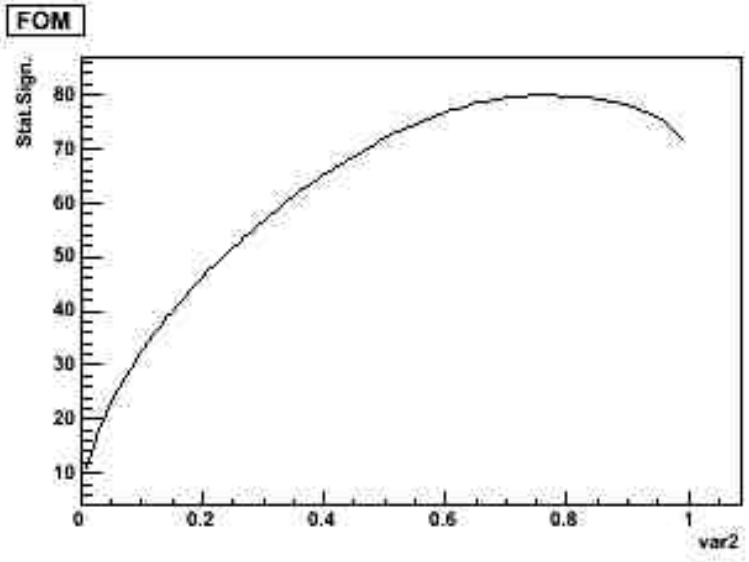
# Esercizio

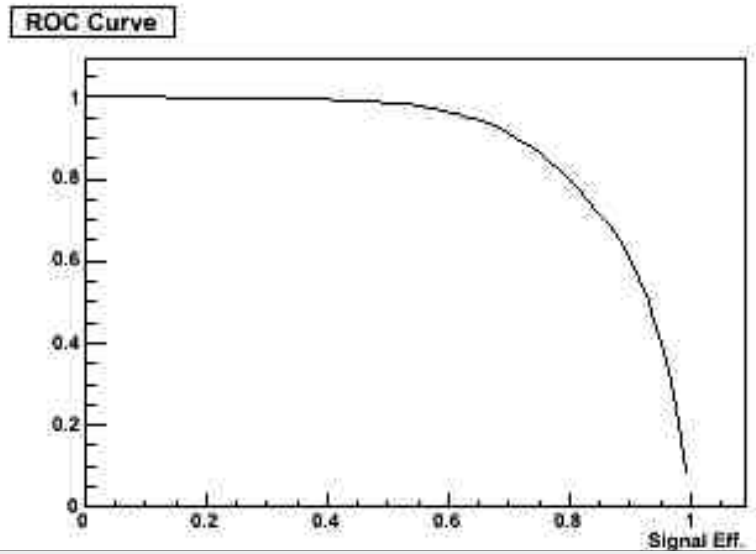
Producete i quattro grafici per $var_1$ e $var_2$



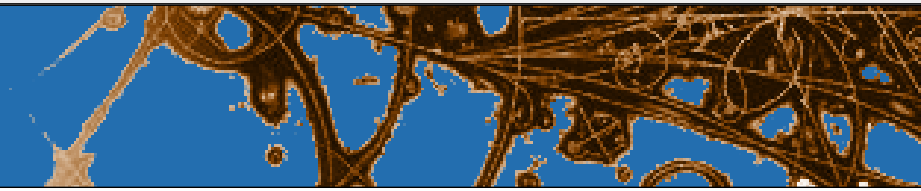
# Introduzione



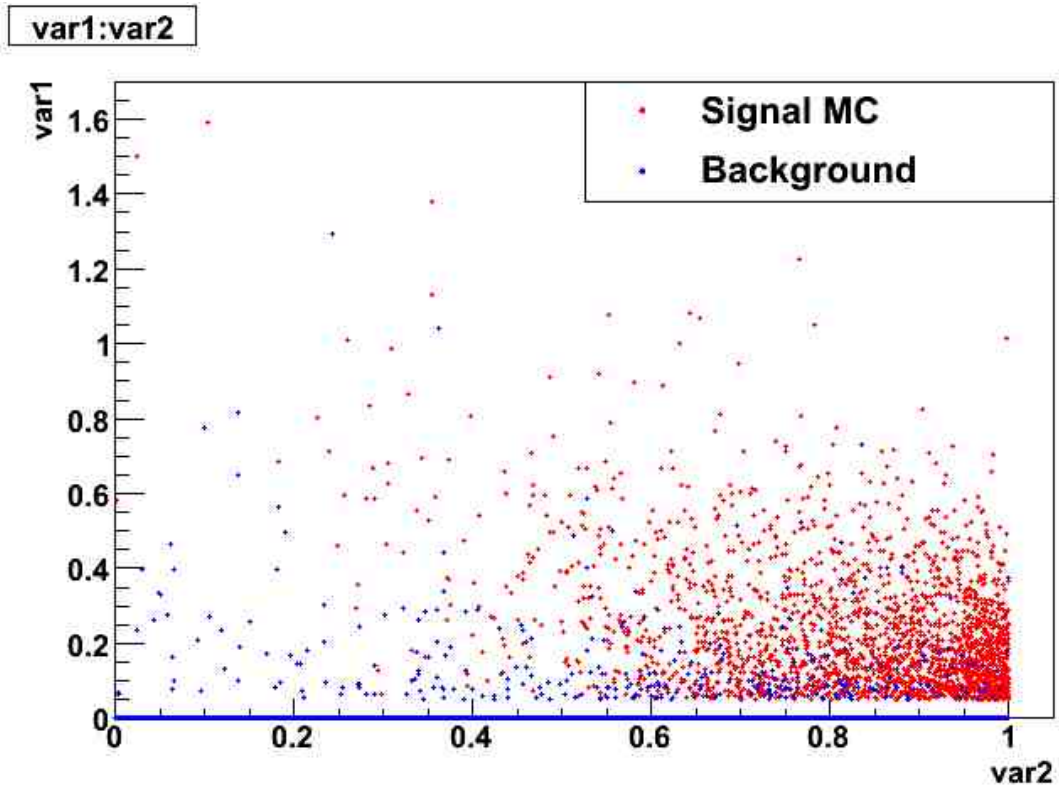



# Outline



# Introduzione

Una cosa interessante e' provare a ottimizzare un taglio nel piano $var_1 \times var_2$




probabilmente il nostro taglio sarà una box del tipo $[0, var_1^{cut}] \times [0, var_2^{cut}]$

# Strategie di Ottimizzazione

## Scan

Una prima strategia di ottimizzazione è possibile grazie all'*ansatz* sulla forma della box di segnale $[0, var_1^{max}] \times [0, var_2^{max}]$.

Possiamo infatti fare uno scan sui possibili valori di $var_{(1,2)}$ nel range $[0, var_{(1,2)}^{max}]$.

Per ogni punto possiamo calcolare la nostra FOM e scegliere come coppia $(var_1^{cut}, var_2^{cut})$ quella per cui la FOM è massima.
La possibilità di fare questo scan in modo semplice è data dal fatto che stiamo variando solo uno dei due estremi della box di segnale.

# Strategie di Ottimizzazione

## Monte Carlo

Una strategia di ottimizzazione alternativa è di fare uno scan "*casuale*". Con un generatore di numeri casuali estraiamo i quattro valori $(var_{(1,2)}^{lowcut}, var_{(1,2)}^{upcut})$ e calcoliamo la FOM nel box $[var_1^{lowcut}, var_1^{upcut}] \times [var_2^{lowcut}, var_2^{upcut}]$.

- Vantaggi:
  - Concettualmente semplice
  - Facile da essere implementato in $N$ dimensioni ($N > 2$)
  - Complessità computazionale inferiore per $N >> 2$
- Svantaggi:
  - Più lento nel caso $N \sim 2$
  - Difficile stabilire quando fermare la ricerca

Una buona ottimizzazione potrebbe essere fare un primo scan Monte Carlo su tutto il range possibile e poi fare un scan fine in zone dove il metodo Monte Carlo indica un alto valore della FOM.

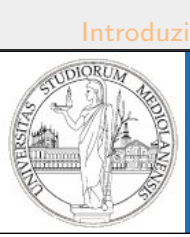
# Strategie di Ottimizzazione

## Bump-Hunter

Il metodo migliore per ottimizzare un taglio a box è il Bump Hunter. Come è possibile implementare tale algoritmo?

1) Ad ogni passaggio restringiamo i quattro lati del box in modo *casuale*.
2) Controlliamo che:
   a) il numero di eventi tagliati sia minore di $N_{tot} * peel$.
   b) FOM nella box sia maggiore della FOM iniziale.
3) Accettiamo la nuova box se le due condizioni sono verificate
4) Reiteriamo fino a che la condizione risulta non soddisfatta per un numero $n_{max}$ di volte consecutive
5) Proviamo ad allargare ciascun taglio (di una quantità piccola) e acettiamo la nuova box se la FOM cresce (*pasting*).

Non ho mai provato a implementare un Bump Hunter (ma lo farò per esercizio!). Un idea utile potrebbe essere generare lo spostamento del punto 1) con una distribuzione gaussiana e non uniforme.



# Esercizio

Ottimizzate il taglio 2D $var_1 \times var_2$ usando un Bump Hunter.