ALMA MATER STUDIORUM - UNIVERSITA' DI BOLOGNA
CAMPUS DI CESENA
SCUOLA DI SCIENZE

CORSO DI LAUREA IN SCIENZE E TECNOLOGIE INFORMATICHE

TITOLO DELLA RELAZIONE FINALE

Implementazione Libreria C per il Multiple Testing Correction

Relazione finale in
Algoritmi e Strutture di Dati

Relatore
Pietro di Lena

Presentata da
Samuele Cancellieri

Sessione: II Sessione

Anno Accademico 2014/2015

Dedicata a tutte le persone che mi hanno aiutato e sostenuto durante gli anni, e a quelle che hanno condiviso con me lo studio di milioni di ore di matematica. Grazie a tutte.

Indice

Introduzione

Nell'era genomica moderna, la mole di dati generata dal sequenziamento genetico è diventata estremamente elevata.

L’analisi di dati genomici richiede l’utilizzo di metodi di significatività statistica per quantificare la robustezza delle correlazioni individuate nei dati.

La significatività statistica ci permette di capire se le relazioni nei dati che stiamo analizzando abbiano effettivamente un peso statistico, cioè se l’evento che stiamo analizzando è successo “per caso” o è effettivamente corretto pensare che avvenga con una probabilità utile.

Indipendentemente dal test statistico utilizzato, in presenza di test multipli di verifica (“Multiple Testing Hypothesis”) è necessario utilizzare metodi per la correzione della significatività statistica (“Multiple Testing Correction”).

Lo scopo di questa tesi è quello di rendere disponibili le implementazioni dei più noti metodi di correzione della significatività statistica.

È stata creata una raccolta di questi metodi, sottoforma di libreria, proprio perché nel panorama bioinformatico moderno non è stato trovato nulla del genere.

La creazione della raccolta di questi metodi è avvenuta in diverse fasi.

La prima fase è stata la cernita dei metodi da includere nella raccolta.

Tra tutti i metodi disponibili ne sono stati scelti 8, considerati i metodi più utilizzati dalla bioinformatica moderna per questo tipo di lavoro.

La seconda fase è stata la scelta del linguaggio di programmazione in cui sviluppare la libreria.

Alla fine la libreria è stata sviluppata in linguaggio C, la scelta è ricaduta su questo linguaggio perché è stato considerato il migliore per prestazioni e portabilità.

La terza fase è stata la vera e propria implementazione degli algoritmi di correzione in linguaggio C.

La quarta fase è stata quella di testing della libreria implementata, necessaria per essere sicuri che i metodi implementati funzionassero correttamente.

La tesi è articolata in quattro capitoli.

Il primo capitolo definisce matematicamente il p-value, il valore che ci permette di effettuare le correzioni statistiche tramite i metodi implementati nella libreria della tesi.

Il secondo capitolo definisce matematicamente i metodi implementati nella libreria della tesi.

La loro formulazione matematica, il tipo di errore statistico che viene corretto (FDR,FWER,QVALUE) e come viene effettuata questa correzione.

Il terzo capitolo mostra come i metodi di correzione sono stati implementati nella libreria.

Il quarto capitolo riassume lo scopo della tesi e come esso è stato raggiunto.

Capitolo 1

Definizione Matematica di P-Value

In questo capitolo verrà definito matematicamente e con linguaggio naturale cos'è il p-value, lo strumento utilizzato dagli algoritmi implementati per eseguire le correzioni statistiche.

Partiamo definendo come sono strutturati i test statistici che utilizzano il p-value.

I test statistici di verifica d'ipotesi sono test che ci permettono di capire se la nostra ipotesi nulla, cioè l'ipotesi che noi riteniamo vera, lo è veramente, cioè se i dati raccolti ci danno un riscontro e confermano la nostra ipotesi.

Ovviamente, visto che dobbiamo fare un confronto, abbiamo anche bisogno di una ipotesi alternativa, cioè l'ipotesi che riteniamo sia l'avvenimento più plausibile che si verifichi nel caso in cui l'ipotesi nulla si rivelasse falsa.

Per fare questo abbiamo bisogno di utilizzare il p-value, che è la probabilità di ottenere un risultato simile o più estremo di quello osservato.

Il p-value ci dice quanto la nostra ipotesi nulla sia vera, cioè quanto è "forte" la nostra ipotesi.

Ovviamente, il p-value viene testato utilizzando una "soglia" che ci permette di capire quanto è robusto il risultato sulle nostre osservazioni, cioè quanto "pesa" sui nostri risultati.

In parole più semplici, ci permette di capire se l'ipotesi nulla è da rigettare sicuramente oppure se possiamo accettarla come vera.

Ora definiremo in modo più rigoroso il p-value.

Matematicamente il p-value viene definito come:

$$Pr(T(yrep)>T(y)|H)$$

Come possiamo vedere è una probabilità, dove yrep rappresenta una replica del test sotto condizione H e T rappresenta il test statistico eseguito.

In altre parole, la probabilità che avvenga un determinato evento durante il test replicato deve essere maggiore rispetto alla probabilità che avvenga

quell'evento ponendo H come condizione, cioè deve assumere un valore "più estremo" rispetto ai dati osservati.

Per valore "più estremo" si intende che i risultati ottenuti non sono contenuti all'interno di una finestra di valori considerati probabili, sono quindi esterni a questo insieme e sono quindi da considerarsi significativi a livello statistico per il fenomeno che stiamo analizzando.

Naturalmente per poter definire se un p-value è uguale o più estremo rispetto allo spazio di valori plausibili è necessario utilizzare un valore di significatività o soglia, solitamente indicato con la lettera greca α

Questo valore è inizialmente arbitrario, cioè viene posto ad un valore considerato giusto per l'analisi che si intende fare, poi viene successivamente corretto seguendo l'andamento dell'osservazione.

Dopo aver selezionato questo valore soglia, possiamo utilizzarlo per testare i nostri p-value, se anche un solo p-value è minore o uguale ad α allora significa che l'ipotesi che abbiamo accettato come vera probabilmente non è corretta ed è quindi da rigettare.

Anche il p-value, che è una funzione di χ, è una variabile casuale, definita nell'intervallo [0,1], assumendo che χ sia continuo, quindi il p-value non è fissato.

Considerando che il p-value non è un valore fissato è impossibile calcolare la sua frequenza, in parole semplici, siccome il p-value è casuale e non fisso, se noi ripetessimo lo stesso test e utilizzassimo la stessa ipotesi tutte le volte, otterremmo sempre dei p-value differenti.

Il valore soglia α può essere considerato come la percentuale di errato rigetto dell'ipotesi nulla, cioè dell'ipotesi definita come vera all'inizio dell'osservazione.

Questo significa che, nel caso utilizzassimo dei p-value definiti entro un insieme conosciuto di valori, cioè se i nostri p-value avessero un valore definito, e permettessimo alla soglia α di assumere valori nell'insieme [0,1], potremmo ottenere una interpretazione dei p-value in termini di α.

Potremmo cioè definire qual è il valore minore di α con cui i p-value ci portano a rigettare l'ipotesi nulla.

Ovviamente potremmo arrivare ad ottenere un valore α che ci permetta di non rigettare l'ipotesi nulla per nessun p-value.

Facciamo un esempio per chiarire meglio il concetto [1].

Uno studio ci dice che circa il 90% dei malati di tumore ai polmoni muore in 3 anni.

Grazie ai nuovi trattamenti questa percentuale è stata ridotta.

Vogliamo dimostrare che la riduzione della mortalità sia effettivamente statisticamente significativa.

Nella ricerca sono stati utilizzati 150 pazienti malati (M), dopo 3 anni ne sono morti 128.

Vogliamo vedere se con una significatività del 5% (soglia $\alpha=0.05$) possiamo concludere che la mortalità del tumore ai polmoni è diminuita.

Facciamo una proporzione per ottenere la probabilità,

$$P=128/150=0.853$$

Ora decidiamo l'ipotesi nulla(H0) e l'ipotesi alternativa(H1):

$$H0=0.90$$

$$H1<0.90.$$

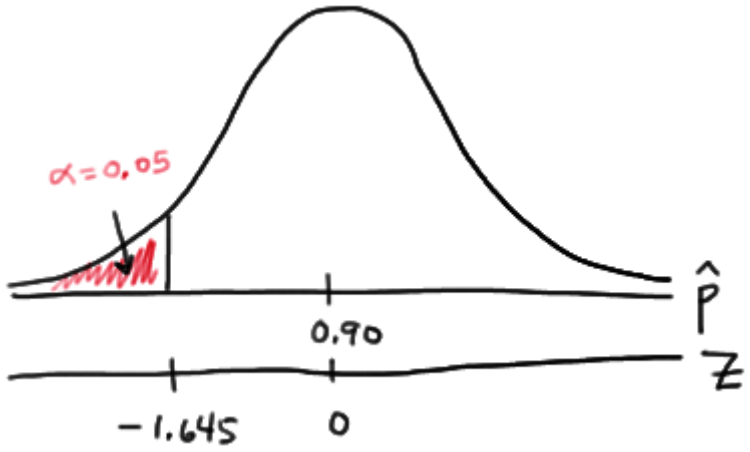
Utilizziamo il seguente algoritmo per effettuare il test statistico(Z-Test).

$$Z=(P-H0)/(\sqrt{H0(1-H0)/M},$$

$$(0.853-0.90)/\sqrt{((0.90*0.10)/150)=-1.92.}$$

Il singolo valore Z non ci dice quanto la riduzione di mortalità al 85% rispetto al 90% sia effettivamente significativa.

La regione di rifiuto è la seguente:



Siccome

$$Z=-1.92<-1.645(\text{regione di } \alpha)$$

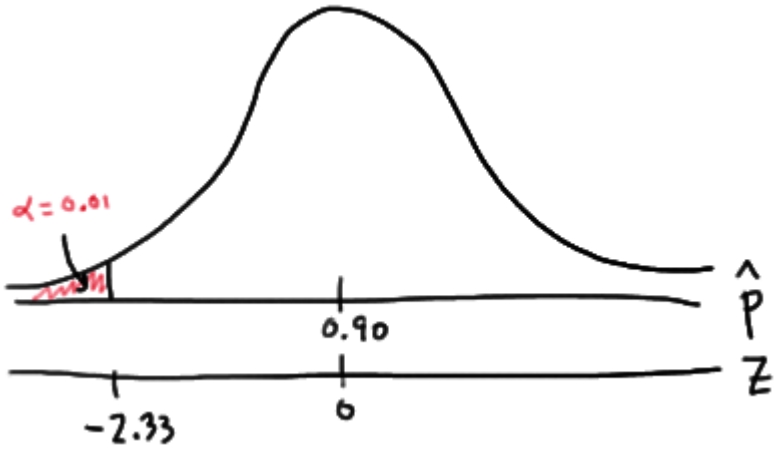
rifiutiamo l'ipotesi nulla e accettiamo l'ipotesi alternativa.

Accettiamo quindi l'ipotesi che ci dice che la mortalità dei tumori polmonari è diminuita con l'uso di nuove tecniche di cura.

Ora vediamo lo stesso esempio con un valore di soglia $\alpha=0.01$.

Poniamo le stesse condizioni e usiamo gli stessi dati.

In questo caso la regione di rifiuto è la seguente:



Siccome

$$Z=-1.92>-2.33(\text{regione di } \alpha)$$

accettiamo l'ipotesi nulla come vera.

In questo caso non abbiamo sufficienti prove a favore dell'ipotesi alternativa, quindi non possiamo dire che le nuove tecniche di cura dei tumori polmonari ne hanno ridotto la mortalità.

Riassumendo possiamo quindi dire che il p-value ci permette di capire se, stando alle nostre osservazioni, è possibile confermare o confutare un'ipotesi che poniamo come vera in partenza.

Con questo esempio molto semplice abbiamo chiarito cos'è il p-value e come si usa, inoltre è stato chiarito cos'è il valore soglia α.

Ora che è chiaro il significato di p-value, bisogna chiarire il perché oltre al semplice p-value si rende necessario l'utilizzo di algoritmi di correzione.

Il motivo è semplice, per evitare che il susseguirsi di test e analisi dei dati renda i dati compromessi, cioè che modifichi l'effettiva significatività dei dati.

In parole semplici, per evitare che la ripetizione dello stesso test un alto numero di volte modifichi la probabilità che l'evento considerato accada.

Facciamo un esempio.

C'è un uomo che durante una tempesta deve attraversare un campo.

Assumiamo che la probabilità che venga colpito da un fulmine nel tragitto, sia del 10%.

È immediato pensare che meno volte il nostro uomo attraversa il campo meno è probabile che si verifichi l'evento del fulmine.

Se infatti il nostro uomo continua ad attraversare il campo, aumenta drasticamente la probabilità che egli venga colpito da un fulmine.

Questa considerazione vale per un qualunque test statistico.

Se ripetiamo più volte lo stesso test rendiamo sempre più probabile l'evento cercato.

Eseguire la correzione ci permette di evitare questo fenomeno, che renderebbe statisticamente inutili i nostri dati.

Capitolo 2

Descrizione Algoritmi Implementati

In questo capitolo verranno descritti in dettaglio gli algoritmi implementati all'interno della libreria.

Gli algoritmi implementati sono 8 e sono algoritmi largamente utilizzati in ambito bioinformatico per effettuare il "Multiple Testing Correction".

Si dividono in diverse categorie, differenziate per il tipo di correzione che applicano al p-value in ingresso e per il tipo di errore che viene analizzato.

Facciamo una breve spiegazione che permetta di capire quale tipo di errore viene valutato dagli algoritmi implementati nella libreria.

Esistono due tipi di errori analizzabili dagli algoritmi di Multiple Testing Correction, gli errori di Tipo I e gli errori di Tipo II.

L'errore di tipo I è conosciuto anche come il "falso positivo", mentre quello di tipo II è conosciuto anche come "falso negativo".

Praticamente gli errori di Tipo I si verificano quando la nostra ipotesi nulla è vera, ma viene rigettata, mentre gli errori di Tipo II si verificano quando la nostra ipotesi nulla dovrebbe essere rigettata, ma non viene fatto.

Gli algoritmi implementati nella libreria andranno ad analizzare solamente errori di Tipo I, anche se utilizzeranno metodi diversi di valutazione.

I diversi metodi di valutazione sono due, il "Family-wise error rate (FWER)" e il "False discovery rate (FDR)".

Il Family-wise error rate (FWER) è definito come la probabilità di ottenere almeno un errore di Tipo I all'interno della nostra correzione, cioè la probabilità che almeno un p-value ci porti a rigettare la nostra ipotesi nulla.

Il False discovery rate (FDR) è definito come la proporzione di falsi positivi (errori di Tipo I) tra le ipotesi rigettate dal test e quelle che pensiamo siano falsi positivi.

Facciamo un esempio per capire meglio.

Se un nostro esperimento ci porta a rigettare 1000 ipotesi e abbiamo un valore FDR di 0.10, possiamo pensare che meno di 100 ipotesi rigettate siano falsi positivi, cioè errori di Tipo I.

2.1 Definizione Errori di Tipo I e Tipo II

In questo paragrafo verrà chiarito meglio il concetto di Errore di Tipo I ed Errore di Tipo II.

Quando decidiamo di effettuare un test statistico ad ipotesi possiamo avere due diversi errori, l'errore di Tipo I, chiamato anche "falso positivo" e l'errore di Tipo II, chiamato anche "falso negativo".

L'errore di Tipo I o falso positivo, consiste nel rifiutare erroneamente un'ipotesi nulla corretta

Abbiamo scelto di rigettare l'ipotesi nulla, l'ipotesi che consideravamo vera all'inizio del test e abbiamo deciso di ritenere quindi vera l'ipotesi alternativa, l'ipotesi più plausibile se decidiamo che quella nulla è in realtà errata.

L'errore di Tipo II o falso negativo, consiste nell'accettare erroneamente un'ipotesi nulla errata.

Abbiamo scelto di accettare l'ipotesi nulla, l'ipotesi che consideravamo vera all'inizio del test e decidiamo di ritenerla vera dopo aver analizzato i risultati, invece di rigettarla ed accettare invece l'ipotesi alternativa, l'ipotesi più plausibile rigettando quella nulla.

Facciamo un semplice esempio per capire come funzionano i due diversi errori.

Analizziamo un caso giuridico.

Le prove raccolte dimostrano che l'imputato è colpevole.

L'ipotesi nulla(H0) che noi consideriamo è : l'imputato è innocente.

Un errore di Tipo I in questo test consisterebbe nel far imprigionare una persona innocente, rigettiamo l'ipotesi H0 e quindi diciamo che l'imputato è colpevole, se poi H0 si rivela essere corretta abbiamo commesso un errore di Tipo I, cioè abbiamo rifiutato un'ipotesi nulla corretta e abbiamo condannato un innocente.

Un errore di Tipo II in questo test consisterebbe nel lasciare libera una persona colpevole, accettiamo l'ipotesi H0 e quindi diciamo che l'imputato è innocente, se poi H0 si rivela essere errata abbiamo commesso un errore di Tipo II, cioè abbiamo accettato un'ipotesi nulla errata e abbiamo lasciato libero un colpevole.

2.2 Metodi di Correzione implementati nella libreria

1) Metodo Bonferroni (Anni ’60) Tipo Correzione: FWER
2) Metodo Sidak (1967) Tipo Correzione: FWER
3) Metodo Sidak-Stepdown (Fine Anni ’60) Tipo Correzione: FWER
4) Metodo Holm (1979) Tipo Correzione: FWER
5) Metodo Hochberg (1988) Tipo Correzione: FWER
6) Metodo Benjamini-Hochberg (1995) Tipo Correzione: FDR
7) Metodo Benjamini-Hochberg-Yekutieli (2001) Tipo Correzione: FDR
8) Metodo Storey-Tibshirani (2003) Tipo Correzione: FDR(Q-Value)

2.2.1 Metodo Bonferroni

Il primo metodo utilizza la famosa Correzione Bonferroni [2], attribuita al matematico italiano Carlo Emilio Bonferroni (1892-1960).

Il metodo di correzione oggi noto con il nome di “correzione di Bonferroni”, è stato utilizzato per la prima volta negli anni ’60 dal matematico Olive Jean Dunn [3][4].

Questo metodo utilizza il FWER come sistema di controllo errori.

La formulazione matematica del metodo è la seguente:

$$pcorretto = pvalue * len$$

con len=numero di p-value inseriti dall’utente,

$$pcorretto = pcorretto <= \alpha$$

con α = valore soglia impostata dall’utente.

Il metodo è molto semplice ed implementabile efficientemente.

In termini pratici, la funzione riceve in ingresso un array contente i p-value da analizzare, ogni singolo p-value viene moltiplicato per la lunghezza dell’array (cioè per il numero di p-value dati in input) e poi sottoposto ad un controllo di soglia, cioè ogni p-value deve essere minore o uguale alla soglia fornita in input dall’utente.

Se un p-value supera questa soglia il p-value corretto, corrispondente a quello in input, viene posto ad 1 in modo che l’utente possa capire che l’ipotesi nulla per quel p-value va rigettata.

Qui possiamo vedere un esempio di dati corretti con il metodo Bonferroni:

Input: Corretti:

PVALUE	BONFER
0.007261893222	1.000000000000
0.008050988716	1.000000000000
0.000000647893	0.000019436795
0.000665844349	0.019975330457
0.004267838925	1.000000000000
0.006120031409	1.000000000000
0.003165375420	0.094961262585
0.001242918708	0.037287561240
0.004319692927	1.000000000000
0.001891879007	0.056756370215
0.005683627676	1.000000000000
0.012973260727	1.000000000000
0.004503148005	1.000000000000
0.011249709908	1.000000000000
0.001762104911	0.052863147321
0.015330040136	1.000000000000
0.011462820532	1.000000000000
0.011280010292	1.000000000000
0.001520912080	0.045627362395
0.014436314048	1.000000000000
0.015685976602	1.000000000000
0.078436361988	1.000000000000
0.039636676032	1.000000000000
0.082833605345	1.000000000000
0.067277326652	1.000000000000
0.089793066676	1.000000000000
0.015990579043	1.000000000000
0.092640779723	1.000000000000
0.013269925170	1.000000000000
0.020874485569	1.000000000000

In questo esempio abbiamo 30 p-value di input, seguendo la formula del metodo vengono moltiplicati ad uno ad uno per la lunghezza dell'array in ingresso (cioè 30) e vengono testati per una soglia $\alpha=0.10$.

Come possiamo notare prima di effettuare la correzione, tutti i p-value sono sotto la soglia $\alpha=0.10$, mentre dopo la correzione pochissimi rimangono sotto questa soglia.

Tutti gli altri vengono posti ad 1.

2.2.2 Metodo Sidak

Il secondo metodo analizzato è il metodo Sidak [5], proposto dal matematico Zbyněk Šidák alla fine degli anni ’60.

Questo metodo utilizza il FWER come sistema di controllo errori.

La formulazione matematica del metodo è la seguente:

$$pcontrollo=1-(1-pvalue)^{len}$$

con len=numero di p-value inseriti dall’utente,

pcontrollo=valore che viene sottoposto al controllo di soglia α,

$$pcorretto=pcontrollo<=\alpha$$

con α=valore soglia impostata dall’utente.

Il metodo è semplice, i valori dei p-value dati in input dall’utente vengono sottoposti ad un calcolo matematico, utilizzando la formula soprastante.

Dopo l’applicazione di questo calcolo ogni p-value viene sottoposto ad un controllo di soglia, con un valore fornito in input.

Quando un p-value non rispetta questa soglia, cioè è maggiore della soglia, il p-value corretto, corrispondente a quello analizzato, viene posto ad 1 in modo che l’utente possa capire che l’ipotesi nulla per quel p-value va rigettata.

Qui possiamo vedere un esempio di dati corretti con il metodo Sidak:

Input: Corretti:

PVALUE	SISING
0.007261893222	1.000000000000
0.008050988716	1.000000000000
0.000000647893	0.000019436613
0.000665844349	0.019783666923
0.004267838925	1.000000000000
0.006120031409	1.000000000000
0.003165375420	0.090728795503
0.001242918708	0.036623283533
0.004319692927	1.000000000000
0.001891879007	0.055226559862
0.005683627676	1.000000000000
0.012973260727	1.000000000000
0.004503148005	1.000000000000
0.011249709908	1.000000000000
0.001762104911	0.051534418264
0.015330040136	1.000000000000
0.011462820532	1.000000000000
0.011280010292	1.000000000000
0.001520912080	0.044635270036
0.014436314048	1.000000000000
0.015685976602	1.000000000000
0.078436361988	1.000000000000
0.039636676032	1.000000000000
0.082833605345	1.000000000000
0.067277326652	1.000000000000
0.089793066676	1.000000000000
0.015990579043	1.000000000000
0.092640779723	1.000000000000
0.013269925170	1.000000000000
0.020874485569	1.000000000000

In questo esempio abbiamo 30 p-value di input, seguendo la formula del metodo vengono creati dei valori controllo per ogni pvalue di input, elevando ogni p-value alla lunghezza dell'array in input, che vengono testati per una soglia $\alpha=0.10$.

Come possiamo notare pochi p-value sono sotto questa soglia, un numero identico a quelli corretti con il metodo Bonferroni.

Tutti gli altri vengono posti ad 1.

2.2.3 Metodo Sidak Stepdown

Il terzo metodo implementato è il metodo Sidak-Stepdown [5], che come il nome suggerisce non è altro che un metodo di correzione che si basa sul metodo Sidak con una piccola modifica nel calcolo della correzione dei p-value.

Questo metodo utilizza il FWER come sistema di controllo errori.

La formulazione matematica del metodo è la seguente:

$$pcontrollo=1-(1-pvalue)^{(len-r[i])}$$

con len=numero di p-value inseriti dall'utente,

r[i]=posizione del p-value analizzato,

pcontrollo=valore che viene sottoposto al controllo di soglia α,

$$pcorretto=pcontrollo<=\alpha$$

con α=valore soglia impostata dall'utente.

Dopo l'applicazione di questo calcolo ogni p-value viene sottoposto ad un controllo di soglia, con un valore fornito in input.

Quando un p-value non rispetta questa soglia, cioè è maggiore della soglia, il p-value corretto, corrispondente a quello analizzato, viene posto ad 1 in modo che l'utente possa capire che l'ipotesi nulla per quel p-value va rigettata.

Qui possiamo vedere un esempio di dati corretti con il metodo Sidak Stepdown:

Input: Corretti:

PVALUE	SIDASD
0.007261893222	1.000000000000
0.008050988716	1.000000000000
0.000000647893	0.000019436613
0.000665844349	0.019130560550
0.004267838925	0.093686819961
0.006120031409	1.000000000000
0.003165375420	0.073266739424
0.001242918708	0.034224013426
0.004319692927	0.090844465720
0.001891879007	0.046238626828
0.005683627676	1.000000000000
0.012973260727	1.000000000000
0.004503148005	0.090426683201
0.011249709908	1.000000000000
0.001762104911	0.044819680732
0.015330040136	1.000000000000
0.011462820532	1.000000000000
0.011280010292	1.000000000000
0.001520912080	0.040262899526
0.014436314048	1.000000000000
0.015685976602	1.000000000000
0.078436361988	1.000000000000
0.039636676032	1.000000000000
0.082833605345	1.000000000000
0.067277326652	1.000000000000
0.089793066676	1.000000000000
0.015990579043	1.000000000000
0.092640779723	0.092640779723
0.013269925170	1.000000000000
0.020874485569	1.000000000000

In questo esempio abbiamo 30 p-value di input, seguendo la formula del metodo vengono creati dei valori controllo per ogni pvalue di input, elevando a potenza ogni p-value con la lunghezza dell'array di input(cioè 30) e sottraendo da essa la posizione del p-value preso in esame.

Dopo questa operazione, i valori di controllo vengono testati per una soglia α=0.10.

Come possiamo notare, pochi p-value sono sotto questa soglia, anche se in numero maggiore rispetto alla versione single-step dello stesso metodo.

Tutti gli altri vengono posti ad 1.

2.2.4 Metodo Holm

Il quarto metodo implementato è il metodo Holm [6], proposto alla fine degli anni ‘60 dal matematico Sture Holm.

Questo metodo utilizza il FWER come sistema di controllo errori.

La formulazione matematica del metodo è la seguente:

$$pcontrollo=(len-r[i])*p\text{-}value$$

con len=numero di p-value inseriti dall’utente,

r[i]=posizione del p-value analizzato,

pcontrollo=valore che viene sottoposto al controllo di soglia α,

$$pcorretto=pcontrollo<=\alpha$$

con α=valore soglia impostata dall’utente.

Questo metodo, tramite la formula soprastante, crea come negli algoritmi precedenti un valore di controllo, utilizzando il p-value analizzato.

Dopo l’applicazione di questo calcolo ogni pcontrollo viene sottoposto ad un controllo di soglia, con un valore fornito in input.

Quando il controllo non rispetta questa soglia, cioè è maggiore della soglia, il p-value corretto, corrispondente a quello analizzato, viene posto ad 1 in modo che l’utente possa capire che l’ipotesi nulla per quel p-value va rigettata.

Qui possiamo vedere un esempio di dati corretti con il metodo Holm:

Input: Corretti:

PVALUE	HOLMSD
0.007261893222	1.000000000000
0.008050988716	1.000000000000
0.000000647893	0.000019436795
0.000665844349	0.019309486109
0.004267838925	0.098160295281
0.006120031409	1.000000000000
0.003165375420	0.075969010068
0.001242918708	0.034801723824
0.004319692927	0.095033244386
0.001891879007	0.047296975179
0.005683627676	1.000000000000
0.012973260727	1.000000000000
0.004503148005	0.094566108098
0.011249709908	1.000000000000
0.001762104911	0.045814727678
0.015330040136	1.000000000000
0.011462820532	1.000000000000
0.011280010292	1.000000000000
0.001520912080	0.041064626156
0.014436314048	1.000000000000
0.015685976602	1.000000000000
0.078436361988	1.000000000000
0.039636676032	1.000000000000
0.082833605345	1.000000000000
0.067277326652	1.000000000000
0.089793066676	1.000000000000
0.015990579043	1.000000000000
0.092640779723	0.092640779723
0.013269925170	1.000000000000
0.020874485569	1.000000000000

In questo esempio abbiamo 30 p-value di input, seguendo la formula del metodo vengono creati dei valori controllo per ogni pvalue di input, moltiplicando ogni p-value per la lunghezza dell'array di input e sottraendo ogni volta alla lunghezza la posizione del p-value esaminato.

Dopo questa operazione, i valori di controllo vengono testati per una soglia α=0.10.

Come possiamo notare pochi p-value sono sotto questa soglia, sono in numero uguale a quelli corretti con il metodo Sidak Stepdown.

Tutti gli altri vengono posti ad 1.

2.2.5 Metodo Hochberg

Il quinto metodo è il metodo Hochberg [7], definito il metodo Bonferroni Step-Up, perché è una rielaborazione del metodo di correzione di Bonferroni, elaborato dal matematico israeliano Yosef Hochberg a fine anni ‘80.

Questo metodo utilizza il FWER come sistema di controllo errori.

La formulazione matematica del metodo è la seguente:

$$pcontrollo=(len-r[i])*p\text{-}value$$

con len=numero di p-value inseriti dall’utente,

r[i]=posizione del p-value analizzato,

pcontrollo=valore che viene sottoposto al controllo di soglia α,

$$pcorretto=pcontrollo<=\alpha$$

con α=valore soglia impostata dall’utente.

Questo metodo, tramite la formula soprastante, crea come negli algoritmi precedenti un valore di controllo, utilizzando il p-value analizzato.

Dopo l’applicazione di questo calcolo ogni pcontrollo viene sottoposto ad un controllo di soglia, con un valore fornito in input.

Quando il controllo non rispetta questa soglia, cioè è maggiore della soglia, il p-value corretto, corrispondente a quello analizzato, viene posto ad 1 in modo che l’utente possa capire che l’ipotesi nulla per quel p-value va rigettata.

Qui possiamo vedere un esempio di dati corretti con il metodo Hochberg:

Input: Corretti:

PVALUE	HOCHSU
0.007261893222	1.000000000000
0.008050988716	1.000000000000
0.000000647893	0.000019436795
0.000665844349	0.019309486109
0.004267838925	0.098160295281
0.006120031409	1.000000000000
0.003165375420	0.075969010068
0.001242918708	0.034801723824
0.004319692927	0.095033244386
0.001891879007	0.047296975179
0.005683627676	1.000000000000
0.012973260727	1.000000000000
0.004503148005	0.094566108098
0.011249709908	1.000000000000
0.001762104911	0.045814727678
0.015330040136	1.000000000000
0.011462820532	1.000000000000
0.011280010292	1.000000000000
0.001520912080	0.041064626156
0.014436314048	1.000000000000
0.015685976602	1.000000000000
0.078436361988	1.000000000000
0.039636676032	1.000000000000
0.082833605345	1.000000000000
0.067277326652	1.000000000000
0.089793066676	1.000000000000
0.015990579043	1.000000000000
0.092640779723	0.092640779723
0.013269925170	1.000000000000
0.020874485569	1.000000000000

In questo esempio abbiamo 30 p-value di input, seguendo la formula del metodo vengono creati dei valori controllo per ogni pvalue di input, moltiplicando ogni p-value per la lunghezza dell'array di input e sottraendo ogni volta alla lunghezza la posizione del p-value esaminato.

Dopo questa operazione, i valori di controllo vengono testati per una soglia $\alpha=0.10$.

Come possiamo notare pochi p-value sono sotto questa soglia, sono in numero uguale a quelli corretti con il metodo Holm, infatti questo metodo esegue lo stesso tipo di correzione, eseguendo però un'operazione che parte dal valore

minore al maggiore dei p-value, contrariamente a quanto accade nel metodo Holm, dove l'operazione è eseguita in ordine inverso.

Tutti gli altri vengono posti ad 1.

2.2.6 Metodo Benjamini-Hochberg

Il sesto metodo implementato è il metodo Benjamini-Hochberg [8].

Questo metodo è stato progettato da Yosef Hochberg e Yoav Benjamini, due matematici israeliani, a metà degli anni ‘90.

Questo metodo utilizza FDR come sistema di controllo errori.

La formulazione matematica del metodo è la seguente:

$$pcontrollo=(p\text{-}value*len)/r[i]$$

con len=numero di p-value inseriti dall’utente,

r[i]=posizione del p-value analizzato,

pcontrollo=valore che viene sottoposto al controllo di soglia α,

$$pcorretto=pcontrollo<=\alpha$$

con α=valore soglia impostata dall’utente.

Questo metodo, tramite la formula soprastante, crea come negli algoritmi precedenti un valore di controllo, utilizzando il p-value analizzato.

Dopo l’applicazione di questo calcolo ogni pcontrollo viene sottoposto ad un controllo di soglia, con un valore fornito in input.

Quando il controllo non rispetta questa soglia, cioè è maggiore della soglia, il p-value corretto, corrispondente a quello analizzato, viene posto ad 1 in modo che l’utente possa capire che l’ipotesi nulla per quel p-value va rigettata.

Qui possiamo vedere un esempio di dati corretti con il metodo Benjamini-Hochberg:

Input: Corretti:

PVALUE	BENHOC
0.007261893222	0.016758215127
0.008050988716	0.017252118677
0.000000647893	0.000019436795
0.000665844349	0.009987665229
0.004267838925	0.016004395970
0.006120031409	0.015300078523
0.003165375420	0.013565894655
0.001242918708	0.012429187080
0.004319692927	0.014398976422
0.001891879007	0.009459395036
0.005683627676	0.015500802753
0.012973260727	0.021622101212
0.004503148005	0.013509444014
0.011249709908	0.022499419815
0.001762104911	0.010572629464
0.015330040136	0.021900057337
0.011462820532	0.020228506820
0.011280010292	0.021150019297
0.001520912080	0.011406840599
0.014436314048	0.021654471071
0.015685976602	0.021389968094
0.078436361988	0.087151513320
0.039636676032	0.047564011238
0.082833605345	0.088750291441
0.067277326652	0.077627684599
0.089793066676	0.092889379320
0.015990579043	0.020857277012
0.092640779723	0.092640779723
0.013269925170	0.020952513427
0.020874485569	0.026093106962

In questo esempio abbiamo 30 p-value di input, seguendo la formula del metodo vengono creati dei valori controllo per ogni pvalue di input, moltiplicando ogni p-value per la lunghezza dell'array di input e dividendo ogni volta il risultato per la posizione del p-value esaminato.

Dopo questa operazione, i valori di controllo vengono testati per una soglia $\alpha=0.10$.

Come possiamo notare tutti i p-value sono sotto questa soglia, contrariamente a quello che accade nel metodo originale.

2.2.7 Metodo Benjamini-Hocberg-Yekutieli

Il settimo metodo programmato è un'evoluzione del metodo Benjamini-Hochberg [8].

Questo metodo è stato progettato da Daniel Yekutieli e Yoav Benjamini, due matematici israeliani, nei primi anni 2000 [9].

Questo metodo utilizza FDR come sistema di controllo errori.

La formulazione matematica del metodo è la seguente:

supp=r[i]

sum+=1/supp

con supp=posizione del p-value analizzato,

sum=reciproco di ogni supp calcolato,

pcontrollo=(p-value*len*sum)/r[i]

con len=numero di p-value inseriti dall'utente,

r[i]=posizione del p-value analizzato,

pcontrollo=valore che viene sottoposto al controllo di soglia α,

pcorretto=pcontrollo<=α

con α=valore soglia impostata dall'utente.

Questo metodo, tramite la formula soprastante, crea come negli algoritmi precedenti un valore di controllo, utilizzando il p-value analizzato.

Dopo l'applicazione di questo calcolo ogni pcontrollo viene sottoposto ad un controllo di soglia, con un valore fornito in input.

Quando il controllo non rispetta questa soglia, cioè è maggiore della soglia, il p-value corretto, corrispondente a quello analizzato, viene posto ad 1 in modo che l'utente possa capire che l'ipotesi nulla per quel p-value va rigettata.

Qui possiamo vedere un esempio di dati corretti con il metodo Benjamini-Hochberg-Yekutieli:

Input: Corretti:

PVALUE	BEHOYE
0.007261893222	0.066948853770
0.008050988716	0.068921992097
0.000000647893	0.000077649747
0.000665844349	0.039900594056
0.004267838925	0.063937355937
0.006120031409	0.061123616800
0.003165375420	0.054195574566
0.001242918708	0.049654442433
0.004319692927	0.057523725505
0.001891879007	0.037790161435
0.005683627676	0.061925507517
0.012973260727	0.086380016086
0.004503148005	0.053970054982
0.011249709908	0.089884892615
0.001762104911	0.042237518649
0.015330040136	0.087490447229
0.011462820532	0.080812624425
0.011280010292	0.084494054912
0.001520912080	0.045570181397
0.014436314048	0.086509333257
0.015685976602	0.085452647265
0.078436361988	1.000000000000
0.039636676032	1.000000000000
0.082833605345	1.000000000000
0.067277326652	1.000000000000
0.089793066676	1.000000000000
0.015990579043	0.083324553249
0.092640779723	1.000000000000
0.013269925170	0.083705021500
0.020874485569	1.000000000000

In questo esempio abbiamo 30 p-value di input, seguendo la formula del metodo vengono creati dei valori controllo per ogni pvalue di input, moltiplicando ogni p-value per la lunghezza dell'array di input e per un valore sum formato dalla somma dei reciproci delle posizioni dei p-value esaminati, inoltre dividiamo ogni volta il risultato per la posizione del p-value esaminato.

Dopo questa operazione, i valori di controllo vengono testati per una soglia $\alpha=0.10$.

Come possiamo notare meno p-value rispetto al metodo Benjamini-Hochberg sono sotto questa soglia.

Tutti gli altri vengono posti ad 1.

2.2.8 Metodo Storey-Tibshirani

L’ottavo metodo di correzione implementato è il metodo Storey-Tibshirani [10].

Questo metodo è stato sviluppato nel 2003 dai matematici americani John D. Storey e Robert Tibshirani.

Questo metodo utilizza FDR come sistema di controllo errori, ma è molto innovativo perché introduce un nuovo modo di valutare FDR, utilizza infatti il Q-VALUE.

Il Q-VALUE rappresenta la percentuale di FDR che ci dovremmo aspettare sul totale di valori significativi analizzati, per esempio, se decido che i valori significativi sono quelli con un Q-VALUE<=5% significherà che FDR dei valori significativi sarà il 5%.

La formulazione matematica del metodo è la seguente:

pusato=p-value>=lambda

pusato=p-value che rispettano la disequazione,

media-parziale=p-value/n.p-value

media-parziale=media dei p-value usati,

media-totale=p-value/len

media-totale=media dei p-value usati,

pi0=media-totale/media-parziale/(1-lambda)

pi0=valore utilizzato per la creazione dei q-value iniziali.

Questo metodo è molto più complesso dei precedenti, infatti la correzione dei valori avviene in molti più passaggi, uno dei quali è la creazione di un valore pi0 che verrà poi utilizzato nella creazione dei q-value.

Il lambda utilizzato nella formula precedente è un valore deciso dall’utente.

Nel caso in cui lambda non sia settato dall’utente verrà utilizzato un valore predefinito, che corrisponde ad un array di valori.

lambarray=(0,0.90,0.05)

lambarry=array predefinito di lambda che inizia con 0 e termina con 0.90, crescendo con passo 0.05.

Se viene utilizzato questo valore predefinito la creazione di pi0 avviene in un modo diverso, infatti tutti i p-value vengono testati per ogni valore di lambdarray.

pusatoarray=p-value>=lambdarray

pusatoarray=array di p-value che rispettano la disequazione,

media-parziale=p-value/n.p-value

media-parziale=media dei p-value usati,

media-totale=p-value/len

media-totale=media dei p-value usati,

pi0array=media-totale/media-parziale/(1-lambdarray)

pi0array=array di pi0 utilizzati per la creazione dei q-value iniziali.

Se è stato utilizzato il metodo predefinito con lambarray, verrà utilizzata una funzione di spline per determinare un singolo valore di pi0 da utilizzare.

Alla fine il risultato sarà il medesimo, avremo ottenuto un solo valore di pi0, il primo ottenuto da una semplice disequazione, il secondo facendo una disequazione per ogni valore di lambarray, più l'utilizzo di una spline di interpolazione.

Dopo che abbiamo creato questo valore, possiamo procedere alla creazione dei q-value.

q-value=pi0*len*p-value/r[i]

q-valuesign=qvalue<=α

con len=numero di p-value inseriti dall'utente,

r[i]=posizione del p-value analizzato,

q-value=valore che viene sottoposto al controllo di soglia α,

qvaluesign=valore che viene riportato all'utente.

Questo metodo, tramite le formule soprastanti, crea un array di valori da utilizzare per definire il q-value di ogni p-value.

Dopo l’applicazione di questo calcolo ogni q-value viene sottoposto ad un controllo di soglia, con un valore fornito in input.

Quando il controllo non rispetta questa soglia, cioè è maggiore della soglia, il q-value corretto, corrispondente a quello analizzato, viene posto ad 1 in modo che l’utente possa capire che l’ipotesi nulla per quel q-value va rigettata.

Qui possiamo vedere un esempio di dati corretti con il metodo Storey-Tibshirani:

Input: Corretti:

PVALUE	STOREY
0.007261893222	0.002940037742
0.008050988716	0.003026687487
0.000000647893	0.000003409964
0.000665844349	0.001659542989
0.004267838925	0.002370077897
0.006120031409	0.002684224302
0.003165375420	0.002370077897
0.001242918708	0.001659542989
0.004319692927	0.002370077897
0.001891879007	0.001659542989
0.005683627676	0.002684224302
0.012973260727	0.003659171406
0.004503148005	0.002370077897
0.011249709908	0.003548860846
0.001762104911	0.001659542989
0.015330040136	0.003659171406
0.011462820532	0.003548860846
0.011280010292	0.003548860846
0.001520912080	0.001659542989
0.014436314048	0.003659171406
0.015685976602	0.003659171406
0.078436361988	0.015289739179
0.039636676032	0.008344563375
0.082833605345	0.015570226569
0.067277326652	0.013618892035
0.089793066676	0.016252768372
0.015990579043	0.003659171406
0.092640779723	0.016252768372
0.013269925170	0.003659171406
0.020874485569	0.004577738063

In questo esempio abbiamo 30 p-value di input, seguendo la formula del metodo vengono creati dei valori qvalue per ogni pvalue di input, l'operazione è molto lunga ed è spiegata nell'introduzione al metodo.

Dopo questa operazione, i valori di controllo vengono testati per una soglia $\alpha=0.10$.

Come possiamo notare tutti i q-value generati sono sotto questa soglia.

Nessuno viene posto ad 1.

Capitolo 3

Scelte progettuali e codice degli algoritmi implementati

In questo capitolo verranno presentati i codici degli algoritmi implementati nella libreria con i rispettivi codici e le scelte effettuate durante la scrittura della stessa.

Le scelte fatte durante lo sviluppo della libreria sono, per la maggior parte, dovute da esigenze di programmazione, cioè sono state fatte per evitare errori di compilazione e di esecuzione della libreria stessa.

Si è cercato infatti di ottimizzare la velocità della libreria evitando ridondanze e sprechi di memoria.

Di seguito verranno visualizzati e spiegati i metodi implementati nella libreria.

3.1 Risultati ottenuti tramite correzioni

In questo paragrafo vengono mostrati alcuni risultati ottenuti sottoponendo un campione di p-value, creati randomicamente, alle correzioni offerte dalla libreria.

La soglia inserita nel test è 0.05.

I p-value analizzati sono 50.

PVALUE	BONFER	HOCHSU	SISING	BENHOC	BEHOYE	HOLMSD	SIDASD	STOREY
0.005013304886	1.000000000000	1.000000000000	1.000000000000	0.022787749482	1.000000000000	1.000000000000	1.000000000000	0.006597494494
0.007736644492	1.000000000000	1.000000000000	1.000000000000	0.025788814975	1.000000000000	1.000000000000	1.000000000000	0.007815811747
0.009409683821	1.000000000000	1.000000000000	1.000000000000	0.026138010615	1.000000000000	1.000000000000	1.000000000000	0.008131825525
0.000041276836	0.002063841793	0.002063841793	0.002061756043	0.002063841793	0.009285648012	0.002063841793	0.002061756043	0.000642084113
0.008039126654	1.000000000000	1.000000000000	1.000000000000	0.025122252043	1.000000000000	1.000000000000	1.000000000000	0.007815811747
0.004680160449	1.000000000000	1.000000000000	1.000000000000	0.023400802246	1.000000000000	1.000000000000	1.000000000000	0.006597494494
0.004075271475	1.000000000000	1.000000000000	1.000000000000	0.025470446720	1.000000000000	1.000000000000	1.000000000000	0.006597494494
0.009023836739	1.000000000000	1.000000000000	1.000000000000	0.026540696291	1.000000000000	1.000000000000	1.000000000000	0.008131825525
0.003132528198	1.000000000000	1.000000000000	1.000000000000	0.022375201418	1.000000000000	1.000000000000	1.000000000000	0.006597494494
0.001020096732	1.000000000000	0.048964643129	0.049750663387	0.017001612197	1.000000000000	0.048964643129	0.047809000381	0.005111208178
0.007645022719	1.000000000000	1.000000000000	1.000000000000	0.027303652568	1.000000000000	1.000000000000	1.000000000000	0.007815811747
0.002358759810	1.000000000000	1.000000000000	1.000000000000	0.019656331753	1.000000000000	1.000000000000	1.000000000000	0.006115303212
0.002204591647	1.000000000000	1.000000000000	1.000000000000	0.022045916469	1.000000000000	1.000000000000	1.000000000000	0.006115303212
0.000703324527	0.035166226344	0.034462901817	0.034567024293	0.017583113172	1.000000000000	0.034462901817	0.033887533699	0.005111208178
0.006176125219	1.000000000000	1.000000000000	1.000000000000	0.023754327766	1.000000000000	1.000000000000	1.000000000000	0.007390235305
0.001314310674	1.000000000000	1.000000000000	1.000000000000	0.016428883428	1.000000000000	1.000000000000	1.000000000000	0.005111208178
0.004218380858	1.000000000000	1.000000000000	1.000000000000	0.023435449213	1.000000000000	1.000000000000	1.000000000000	0.006597494494
0.670908601801	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000
0.895223539274	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000
0.605911523852	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000
0.899782748846	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000
0.690323399701	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000
0.139867436206	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000
0.070914168409	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	0.035584170528
0.606738928988	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000
0.645325608386	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000
0.479898595475	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000
0.313034667779	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000
0.154862429087	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000
0.029027395895	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	0.022576863474
0.147869391901	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000
0.506752661665	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000
0.488784085255	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000
0.051847356861	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	0.032614835298
0.065775888071	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	1.000000000000	0.035584170528

In questa tabella vengono riportati i risultati della correzione dei p-value (prima colonna).

Notiamo subito che la colonna del metodo Bonferroni (seconda colonna) riporta quasi esclusivamente valori uguali a 1, infatti è uno dei metodi più stringenti implementato nella libreria.

Anche il metodo Benjamini-Hochberg-Yekutieli (sesta colonna) riporta praticamente 49 valori pari ad 1, infatti è anch'esso un metodo estremamente stringente.

Tutti gli altri metodi, ad esclusione dei metodi Storey e Benjamini-Hochberg che sono molto permissivi, riportano moltissimi valori uguali ad 1.

3.2 Metodi di Correzione implementati nella libreria

1) Metodo Bonferroni (Anni ’60) Tipo Correzione: FWER
2) Metodo Sidak (1967) Tipo Correzione: FWER
3) Metodo Sidak-Stepdown (Fine Anni ’60) Tipo Correzione: FWER
4) Metodo Holm (1979) Tipo Correzione: FWER
5) Metodo Hochberg (1988) Tipo Correzione: FWER
6) Metodo Benjamini-Hochberg (1995) Tipo Correzione: FDR
7) Metodo Benjamini-Hochberg-Yekutieli (2001) Tipo Correzione: FDR
8) Metodo Storey-Tibshirani (2003) Tipo Correzione: FDR(Q-Value)

3.2.1 Metodo Bonferroni

Il primo metodo presentato nella libreria è il metodo Bonferroni, come spiegato precedentemente è un metodo che utilizza FWER come controllo dell’errore.

Il codice in C contenuto nella libreria è il seguente:

```
double *bonferroni(double *a,int len,double sign)
{
    int i;
    double *pbonferroni,temp;
    if((pbonferroni=(double*)malloc(len*sizeof(double)))==NULL)
    {
        return NULL;
    }

    for(i=0;i<len;i++)
    {
        temp=a[i]*len;
        if(temp<=sign)
        {
            pbonferroni[i]=temp;
        }
        else
        {
            pbonferroni[i]=1;
        }
    }
    return pbonferroni;
}
```

Il metodo viene richiamato fornendo in input un puntatore a double *a = array dei p-value da correggere, un intero len=lunghezza dell'array in input e un double sign=soglia di controllo decisa dall'utente.

Quando il metodo viene richiamato vengono creati un indice intero i e due double, il primo double (*pbonferroni) è l'array che conterrà i p-value corretti, mentre il secondo double (temp) verrà usato come valore temporaneo per effettuare la correzione.

Dopo la creazione delle variabili, allochiamo la memoria necessaria per salvare i p-value all'interno dell'array pbonferroni.

In seguito viene eseguito un ciclo che scorre tutto l'array dei p-value originali e moltiplica ogni singolo p-value per la lunghezza dell'array di p-value in ingresso, se il p-value corretto rispetta la condizione di essere minore o uguale alla soglia scelta in input viene inserito nell'array dei p-value corretti, tutte le altre posizioni, cioè quelle che non rispettano la condizione della soglia, vengono poste ad 1.

3.2.2 Metodo Sidak

Il secondo metodo presentato nella libreria è il metodo Sidak, come spiegato precedentemente è un metodo che utilizza FWER come controllo dell'errore.

Il codice in C contenuto nella libreria è il seguente:

```
double *sidaksinglestep(double *a,int len,double sign)
{
    int i;
    double *psidak,temp;
    if((psidak=(double*)malloc(len*sizeof(double)))==NULL)
    {
        return NULL;
    }

    for(i=0;i<len;i++)
    {
        temp=1.0-pow(1.0-a[i],len);
        if(temp<=sign)
        {
            psidak[i]=temp;
        }
        else
        {
            psidak[i]=1;
        }
    }
    return psidak;
}
```

Il metodo viene richiamato fornendo in input un puntatore a double *a = array dei p-value da correggere, un intero len=lunghezza dell'array in input e un double sign=soglia di controllo decisa dall'utente.

Quando il metodo viene richiamato vengono creati un indice intero i e due double, il primo double (*psidak) è l'array che conterrà i p-value corretti, mentre il secondo double (temp) verrà usato come valore temporaneo per effettuare la correzione.

Dopo la creazione delle variabili, allochiamo la memoria necessaria per salvare i p-value all'interno dell'array psidak.

In seguito viene eseguito un ciclo che scorre tutto l'array dei p-value originali e eleva ogni singolo p-value alla lunghezza dell'array di p-value in

ingresso, se il p-value corretto rispetta la condizione di essere minore o uguale alla soglia scelta in input viene inserito nell'array dei p-value corretti, tutte le altre posizioni, cioè quelle che non rispettano la condizione della soglia, vengono poste ad 1.

3.2.3 Metodo Sidak Stepdown

Il terzo metodo presentato nella libreria è il metodo Sidak Stepdown, come spiegato precedentemente è un metodo che utilizza FWER come controllo dell'errore.

Il codice in C contenuto nella libreria è il seguente:

```
double *sidakstepdown(double *a,int len,double sign,size_t pindex[],size_t prank[])
{
    int i;
    double *psidaksd,temp,control;

    if((psidaksd=(double*)malloc(len*sizeof(double)))==NULL)
    {
        return NULL;
    }

    for(i=len-1;i>=0;i--)
    {
        temp=(1-a[i]);
        control=(1-(pow(temp,(len-i))));
        if(control<=sign)
        {
            psidaksd[pindex[i]]=control;
        }
        else
        {
            psidaksd[pindex[i]]=1;
        }
    }
    return psidaksd;
}
```

Il metodo viene richiamato fornendo in input un puntatore a double *a = array dei p-value da correggere, un intero len=lunghezza dell'array in input, un double sign=soglia di controllo decisa dall'utente, un array size_t pindex=array degli indici dell'array di p-value in input e un array size_t prank=array del rank dell'array di p-value in input.

Quando il metodo viene richiamato vengono creati un indice intero i e tre double, il primo double (*psidaksd) è l'array che conterrà i p-value corretti, il secondo double (temp) verrà usato come valore temporaneo per effettuare la correzione, cosi come il terzo double (control) che verrà anch'esso utilizzato per effettuare la correzione dei p-value.

Dopo la creazione delle variabili, allochiamo la memoria necessaria per salvare i p-value all'interno dell'array psidaksd.

In seguito viene eseguito un ciclo che scorre tutto l'array.

In questo caso però partendo dall'ultimo elemento dei p-value originali e eleva ogni singolo p-value alla lunghezza dell'array di p-value in ingresso, sottraendo ad ogni passaggio la posizione del p-value analizzato.

Se il p-value corretto rispetta la condizione di essere minore o uguale alla soglia scelta in input viene inserito nell'array dei p-value corretti, tutte le altre posizioni, cioè quelle che non rispettano la condizione della soglia, vengono poste ad 1.

3.2.4 Metodo Holm

Il quarto metodo presentato nella libreria è il metodo Holm, come spiegato precedentemente è un metodo che utilizza FWER come controllo dell'errore.

Il codice in C contenuto nella libreria è il seguente:

```
double *holmstepdown(double *a,int len,double sign,size_t pindex[],size_t prank[])
{
    int i;
    double *pholmsd,control;

    if((pholmsd=(double*)malloc(len*sizeof(double)))==NULL)
    {
        return NULL;
    }

    for(i=len-1;i>=0;i--)
    {
        control=(len-i)*a[i];
        if(control<=sign)
        {
            pholmsd[pindex[i]]=control;
        }
        else
        {
           pholmsd[pindex[i]]=1;
        }
    }
    return pholmsd;
}
```

Il metodo viene richiamato fornendo in input un puntatore a double *a = array dei p-value da correggere, un intero len=lunghezza dell'array in input, un double sign=soglia di controllo decisa dall'utente, un array size_t pindex=array degli indici dell'array di p-value in input e un array size_t prank=array del rank dell'array di p-value in input.

Quando il metodo viene richiamato vengono creati un indice intero i e due double, il primo double (*pholmsd) è l'array che conterrà i p-value corretti, il secondo double (control) verrà usato come valore per effettuare la correzione.

Dopo la creazione delle variabili, allochiamo la memoria necessaria per salvare i p-value all'interno dell'array pholmsd.

In seguito viene eseguito un ciclo che scorre tutto l'array.

In questo caso però partendo dall'ultimo elemento, dei p-value originali e moltiplica ogni singolo p-value alla lunghezza dell'array di p-value in ingresso,sottraendo ad ogni passaggio la posizione del p-value analizzato.

Se il p-value corretto rispetta la condizione di essere minore o uguale alla soglia scelta in input viene inserito nell'array dei p-value corretti, tutte le altre posizioni, cioè quelle che non rispettano la condizione della soglia, vengono poste ad 1.

3.2.5 Metodo Hochberg

Il quinto metodo presentato nella libreria è il metodo Hochberg, come spiegato precedentemente è un metodo che utilizza FWER come controllo dell'errore.

Il codice in C contenuto nella libreria è il seguente:

```
double *hochbergstepup(double *a,int len,double sign,size_t pindex[],size_t prank[])
{
    int i;
    double *phocsu,temp;

    if((phocsu=(double*)malloc(len*sizeof(double)))==NULL)
    {
        return NULL;
    }

    for(i=0;i<len;i++)
    {
        temp=(len-i)*a[i];
        if(temp<=sign)
        {
            phocsu[pindex[i]]=temp;
        }
        else
        {
            phocsu[pindex[i]]=1;
        }
    }
    return phocsu;
}
```

Il metodo viene richiamato fornendo in input un puntatore a double *a = array dei p-value da correggere, un intero len=lunghezza dell'array in input, un double sign=soglia di controllo decisa dall'utente, un array size_t pindex=array degli indici dell'array di p-value in input e un array size_t prank=array del rank dell'array di p-value in input.

Quando il metodo viene richiamato vengono creati un indice intero i e due double, il primo double (*phochsu) è l'array che conterrà i p-value corretti, il secondo double (temp) verrà usato come valore temporaneo per effettuare la correzione.

Dopo la creazione delle variabili, allochiamo la memoria necessaria per salvare i p-value all'interno dell'array phochsu.

In seguito viene eseguito un ciclo che scorre tutto l'array dei p-value originali e moltiplica ogni singolo p-value alla lunghezza dell'array di p-value in ingresso,sottraendo ad ogni passaggio la posizione del p-value analizzato.

Se il p-value corretto rispetta la condizione di essere minore o uguale alla soglia scelta in input viene inserito nell'array dei p-value corretti, tutte le altre posizioni, cioè quelle che non rispettano la condizione della soglia, vengono poste ad 1.

3.2.6 Metodo Benjamini-Hochberg

Il sesto metodo presentato nella libreria è il metodo Benjamini-Hochberg, come spiegato precedentemente è un metodo che utilizza FDR come controllo dell'errore.

Il codice in C contenuto nella libreria è il seguente:

```
double *benjaminihochberg(double *a,int len,double sign,size_t pindex[],size_t prank[])
{
    int i;
    double *pbenjaminihoch,control;

    if((pbenjaminihoch=(double*)malloc(len*sizeof(double)))==NULL)
    {
        return NULL;
    }

    for(i=0;i<len;i++)
    {
        control=(a[i]*len)/(i+1);
        if(control<=sign)
        {
            pbenjaminihoch[pindex[i]]=control;
        }
        else
        {
            pbenjaminihoch[pindex[i]]=1;
        }
    }
    return pbenjaminihoch;
}
```

Il metodo viene richiamato fornendo in input un puntatore a double *a = array dei p-value da correggere, un intero len=lunghezza dell'array in input, un double sign=soglia di controllo decisa dall'utente, un array size_t pindex=array degli indici dell'array di p-value in input e un array size_t prank=array del rank dell'array di p-value in input.

Quando il metodo viene richiamato vengono creati un indice intero i e due double, il primo double (*pbenjaminihoch) è l'array che conterrà i p-value corretti, il secondo double (control) verrà usato come valore temporaneo per effettuare la correzione.

Dopo la creazione delle variabili, allochiamo la memoria necessaria per salvare i p-value all'interno dell'array pbenjaminihoch.

In seguito viene eseguito un ciclo che scorre tutto l'array dei p-value originali e moltiplica ogni singolo p-value alla lunghezza dell'array di p-value in ingresso, dividendo il risultato ad ogni passaggio per la posizione del p-value analizzato.

Se il p-value corretto rispetta la condizione di essere minore o uguale alla soglia scelta in input viene inserito nell'array dei p-value corretti, tutte le altre posizioni, cioè quelle che non rispettano la condizione della soglia, vengono poste ad 1.

3.2.7 Metodo Benjamini-Hocberg-Yekutieli

Il settimo metodo presentato nella libreria è il metodo Benjamini-Hochberg-Yekutieli, come spiegato precedentemente è un metodo che utilizza FDR come controllo dell'errore.

Il codice in C contenuto nella libreria è il seguente:

```
double *benjaminihochbergyekutieli(double *a,int len,double sign,size_t pindex[],size_t prank[])
{
    int i;
    double *pbenhocyek,sum,control,supp;

    if((pbenhocyek=(double*)malloc(len*sizeof(double)))==NULL)
    {
        return NULL;
    }

    supp=0;
    sum=0;
    for(i=0;i<len;i++)
    {
        supp=i+1;
        sum+=1/supp;
    }


    for(i=0;i<len;i++)
    {
        control=(a[i]*len*sum)/(i+1);
        if(control<=sign)
        {
            pbenhocyek[pindex[i]]=control;
        }
        else
        {
            pbenhocyek[pindex[i]]=1;
        }
    }
    return pbenhocyek;
}
```

Il metodo viene richiamato fornendo in input un puntatore a double *a = array dei p-value da correggere, un intero len=lunghezza dell'array in input, un double sign=soglia di controllo decisa dall'utente, un array size_t pindex=array degli indici dell'array di p-value in input e un array size_t prank=array del rank dell'array di p-value in input.

Quando il metodo viene richiamato vengono creati un indice intero i e quattro double, il primo double (*pbenhocyek) è l'array che conterrà i p-value corretti, il secondo double (sum) verrà usato come valore per effettuare la

correzione, precisamente sarà la somma dei reciproci delle posizioni dei p-value di input, il terzo double (control) verrà usato per effettuare la correzione, il quarto double (supp) invece è stato introdotto per permettere il calcolo di sum, in quanto il compilatore non effettuava correttamente l'operazione di somma senza una variabile d'appoggio apposita.

Dopo la creazione delle variabili, allochiamo la memoria necessaria per salvare i p-value all'interno dell'array pbenhocyek.

In seguito viene eseguito un ciclo che scorre tutto l'array dei p-value originali e moltiplica ogni singolo p-value per la lunghezza dell'array di p-value in ingresso, dividendo il risultato ad ogni passaggio per la posizione del p-value analizzato, inoltre moltiplica i p-value per la variabile sum.

Se il p-value corretto rispetta la condizione di essere minore o uguale alla soglia scelta in input viene inserito nell'array dei p-value corretti, tutte le altre posizioni, cioè quelle che non rispettano la condizione della soglia, vengono poste ad 1.

3.2.8 Metodo Storey-Tibshirani

L'ottavo metodo presentato nella libreria è il metodo Benjamini-Hochberg-Yekutieli, come spiegato precedentemente è un metodo che utilizza FDR come controllo dell'errore.

Questo algoritmo è stato suddiviso in due funzioni separate.

La prima funzione esegue tutti i passaggi preliminari alla correzione dei p-value.

La seconda funzione esegue invece la correzione dei p-value.

Il codice in C contenuto nella libreria è il seguente:

Prima funzione.

```
double createpi0(int lamblen,int len,double *pvalues,double lamb)
{
    double *pi0array,*lambarray;
    double temp,pi0,pi0mediaparziale,pi0mediatotale,pi0mediafinale;
    double pi0spline[3];
    int contatore,i,cont,j;

     if((pi0array=(double*)malloc(19*sizeof(double)))==NULL)
    {
        return 0;
    }

     if((lambarray=(double*)malloc(19*sizeof(double)))==NULL)
    {
        free(pi0array);
        return 0;
    }

    if(lamblen==1)
    {
    printf("Using assigned lambda\n");
        cont=0;
        temp=0;
        for(i=0;i<len;i++)
        {
            if(pvalues[i]>=lamb)
            {
                temp+=pvalues[i];
                cont++;
            }
        }

        double mediaparziale=temp/cont;
        double mediatotale=temp/len;
        pi0=(mediatotale/mediaparziale)/(1-lamb);
```

```
    if(pi0==0 || isnan(pi0))
    {
        printf("Failed to calculate pi0, use another lambda method\n");
        return 0;
    }
}
else
{
    for(i=0;i<19;i++)
    {
        lambarray[i]=(i*0.05);
        pi0array[i]=0;

    }
    temp=0;
    contatore=0;

    for(i=0;i<19;i++)
    {
        for(j=0;j<len;j++)
        {
            if(pvalues[j]>=lambarray[i])
            {
                contatore++;
                temp+=pvalues[j];
            }
        }

        pi0mediaparziale=(temp/contatore);
        pi0mediatotale=(temp/len);
        pi0mediafinale=(pi0mediatotale/pi0mediaparziale);
        pi0array[i]=(pi0mediafinale/(1-lambarray[i]));

        temp=0;
        contatore=0;
    }
```

```
        //SPLINE

        gsl_interp_accel *acc=gsl_interp_accel_alloc();

        gsl_spline *spline = gsl_spline_alloc(gsl_interp_cspline,19);

        gsl_spline_init(spline,lambarray,pi0array,19);

        pi0spline[0]=gsl_spline_eval(spline,lambarray[19-3],acc);
        pi0spline[1]=gsl_spline_eval(spline,lambarray[19-2],acc);
        pi0spline[2]=gsl_spline_eval(spline,lambarray[19-1],acc);

        for(i=0;i<3;i++)
        {
        pi0+=pi0spline[i];
        }

        pi0=pi0/3-(exp(-6.0));

        if(isnan(pi0) || pi0==0)
        {
            printf("Failed to calculate pi0, use another lambda method\n");
            return 0;
        }

        gsl_spline_free(spline);

        gsl_interp_accel_free(acc);
        //FINE SPLINE
    }
    pi0=MIN(pi0,1);
    printf("pi0 finale: %.12f\n",pi0);
    return pi0;
}
```

Questa prima funzione (createpi0) permette di creare il valore pi0, cioè una stima delle ipotesi nulle dei p-value dati in input.

La funzione createpi0 viene richiamata dalla funzione principale (storey) che analizzeremo successivamente.

La funzione viene richiamata fornendo in input un puntatore a double *pvalues = array dei p-value da correggere, un intero len=lunghezza dell'array in input, un double lamb=valore lambda deciso dall'utente, può essere nullo, un intero lamblen=lunghezza di lambda, se deciso dall'utente sarà 1.

Dopo la creazione delle variabili, allochiamo la memoria necessaria per salvare i valori di lambda all'interno dell'array lambarray e allochiamo la memoria per salvare i valori che useremo per creare pi0 all'interno dell'array pi0array.

In seguito vengono effettuati calcoli atti a definire pi0, che come accennato prima, è una stima di tutte le ipotesi nulle dei p-value di input.

Come primo passaggio la funzione analizza la lunghezza della stringa lambda per capire se l'utente ha fornito un lambda in input o se vuole utilizzare il lambdarray di default.

Successivamente viene calcolato il valore pi0 da utilizzare per effettuare la correzione.

Nel caso in cui l'utente abbia inserito un valore di lambda, quel valore viene utilizzato per calcolare pi0, ogni p-value viene processato in un ciclo for che prende solo i p-value maggiori o uguali al lambda e ne calcola la media parziale, cioè la media considerando i valori utilizzati, e la media totale, cioè la media considerando l'intero array di p-value di input.

Nel caso in cui l'utente non abbia inserito un valore lambda di input viene creato un array di lambda e viene effettuata un'operazione simile a quella precedente, cioè viene calcolata la media totale e parziale dei p-value che rispettano la diseguaglianza, in questo caso però l'operazione viene fatta per ogni valore dell'array lambarray.

Dopo questa operazione l'array pi0array contiene tutti i valori di pi0 calcolati per ogni lambda di lambarray.

Dopo questa fase preliminare si passa alla creazione di un singolo valore pi0, cioè quello che verrà utilizzato dalla funzione di correzione.

Codice C della funzione di correzione:

```c
double *storey(double *a,double lamb,int lamblen,int len,double sign,size_t pindex[],size_t prank[])
{

    double *pi0array,*qvalues,*qvaluessign,*pvalues,*lambarray;
    double pi0;
    int i;

    if((pi0array=(double*)malloc(19*sizeof(double)))==NULL)
    {
        return NULL;
    }

    if((qvalues=(double*)malloc(len*sizeof(double)))==NULL)
    {

        free(pi0array);
        return NULL;
    }

    if((qvaluessign=(double*)malloc(len*sizeof(double)))==NULL)
    {
        free(pi0array);
        free(qvalues);
        return NULL;
    }

    if((pvalues=(double*)malloc(len*sizeof(double)))==NULL)
    {
        free(pi0array);
        free(qvalues);
        free(qvaluessign);
        return NULL;
    }
```

```
 if((lambarray=(double*)malloc(19*sizeof(double)))==NULL)
{
    free(pi0array);
    free(qvalues);
    free(qvaluessign);
    free(pvalues);
    return NULL;
}

for(i=0;i<len;i++)
{
    if(a[i]>=0 && a[i]<=1)
    {
        pvalues[i]=a[i];
    }
    else
    {
        printf("ERROR: P-Values not in range [0,1]");
        return NULL;
    }
}

if (lamb<0 || lamb>=1)
{
    printf("ERROR: Lambda not in range [0,1)");
}

pi0=createpi0(lamblen,len,pvalues,lamb);
```

```
    for(i=len-1;i>=0;i--)
    {
        qvalues[i]=pi0*len*pvalues[i]/prank[i];
    }

    qvalues[pindex[len-1]]=MIN(qvalues[pindex[len-1]],1);

    for(i=len-2;i>=0;i--)
    {
        qvalues[pindex[i]]=MIN(qvalues[pindex[i]],qvalues[pindex[i+1]]);
        qvalues[pindex[i]]=MIN(qvalues[pindex[i]],1);
    }

    for(i=len-1;i>=0;i--)
    {
        if(qvalues[pindex[i]]<=sign)
        {
            qvaluessign[pindex[i]]=qvalues[pindex[i]];
        }
        else
        {
            qvaluessign[pindex[i]]=1;
        }
    }
    return qvaluessign;
}
```

Il metodo viene richiamato fornendo in input un puntatore a double *a = array dei p-value da correggere, un intero len=lunghezza dell'array in input, un double sign=soglia di controllo decisa dall'utente, un array size_t pindex=array degli indici dell'array di p-value in input, un array size_t prank=array del rank dell'array di p-value in input, un double lamb=valore lambda fornito dall'utente(facoltativo) e un intero lamblen=lunghezza di lambda(1 se inserito dall'utente).

Quando il metodo viene richiamato vengono creati un indice intero i e sei double, il primo double (*pi0array) è l'array che conterrà i valori pi0 creati dalla funzione di creazione pi0, spiegata precedentemente, il secondo double (qvalues) conterrà i valori dei qvalues prima del controllo soglia, il terzo double (qvaluessign) conterrà i valori dei q-value dopo il controllo con la soglia, il quarto double (pvalues) contiene i p-value inseriti in ingresso dall'utente, è un array usato come copia dell'originale in modo da non modificare i dati inseriti in

input dall'utente, il quinto double (lambarray) conterrà i valori lambda usati nella creazione di pi0, come spiegato precedentemente e il sesto double (pi0) sarà il valore ottenuto dalla funzione createpi0 spiegata precedentemente.

Dopo la creazione delle variabili, allochiamo la memoria necessaria per salvare i p-value, i q-value e pi0.

Dopo queste operazioni preliminari viene richiamata la funzione createpi0, che crea il valore pi0, come spiegato precedentemente, e lo ritorna al double (pi0) che verrà utilizzato da questo metodo.

Dopo l'operazione di creazione di pi0 viene eseguito un ciclo che scorre l'array di p-value e assegna l'array di q-value, ogni p-value viene moltiplicato per pi0 e per la lunghezza totale dell'array di p-value originale, poi il risultato viene diviso per il rank del p-value in analisi.

In seguito viene eseguito un ciclo che scorre tutto l'array dei q-value, se il q-value rispetta la condizione di essere minore o uguale alla soglia scelta in input viene inserito nell'array dei q-value corretti (qvaluessign), tutte le altre posizioni, cioè quelle che non rispettano la condizione della soglia, vengono poste ad 1.

Conclusioni

L’obiettivo di questa tesi è stato quello di creare una libreria comoda e di semplice utilizzo per il “Multiple Testing Correction”.

L’idea alla base dello sviluppo della libreria è proprio quello di racchiudere in un unico luogo molti algoritmi necessari per la gestione delle informazioni nelle analisi della moderna biologia.

La libreria è stata sviluppata in linguaggio C, ritenuto uno dei migliori linguaggi di programmazione di alto livello, è anche uno dei linguaggi con le migliori prestazioni ed una portabilità eccellente.

Per queste motivazioni è stato scelto come linguaggio per la realizzazione della libreria, nonostante altri alcune implementazioni risulterebbero più semplici in altri linguaggi di programmazione (come, ad esempio, R e Matlab).

Tutto il lavoro svolto nella realizzazione della libreria è stato fatto per rendere di semplice ed immediato utilizzo il pacchetto anche per chi si intende meno di programmazione.
L’obiettivo è infatti quello di fornire un pacchetto pronto all’uso che possa essere facilmente utilizzato e compreso da chi vorrà farne uso.

Il lavoro svolto si è articolato in più fasi, una principale, cioè quella di “traduzione” degli algoritmi matematici in codice C funzionante e una fase secondaria, di testing, in modo che i valori ottenuti dalla correzione dei p-value fossero coerenti con i metodi matematici.

Bibliografia

[1] The P-Value Approach – STAT 414/415 - Eberly College of Science - The Pennsylvania State University

[2] Bonferroni, C. E. (1936) - Teoria statistica delle classi e calcolo delle probabilità - Pubblicazioni del R. Istituto Superiore di Scienze Economiche e Commerciali di Firenze.

[3] Dunn, O. J. (1959) - "Estimation of the Medians for Dependent Variables" - Annals of Mathematical Statistics 30 (1).

[4] Dunn, O. J. (1961) - "Multiple Comparisons Among Means" - Journal of the American Statistical Association 56 (293).

[5] Šidák, Z. K. (1967) - "Rectangular Confidence Regions for the Means of Multivariate Normal Distributions" - Journal of the American Statistical Association 62 (318).

[6] Holm, S. (1979) - "A simple sequentially rejective multiple test procedure" - Scandinavian Journal of Statistics 6 (2).

[7] Hochberg, Y (1988) - "A Sharper Bonferroni Procedure for Multiple Tests of Significance" - Biometrika 75 (4).

[8] Benjamini, Y. and Hochberg, Y. (1995). Controlling the false discovery rate: A practical and powerful approach to multiple testing - J. Roy. Statist. Soc. Ser. B 57 289–300.

[9] Benjamini, Y. and Yekutieli, D. (2001). The control of the false discovery rate in multiple testing under dependency. Ann.Statist. 29 1165–1188.

[10] Storey, John D.; Tibshirani, R. (2003) - "Statistical significance for genome-wide studies" – Proceedings of the National Academy of Sciences **100** (16).