Università di Pisa

Facoltà di Ingegneria

Corso di Laurea Specialistica in Ingegneria Biomedica

TESI DI LAUREA:

IMPLEMENTAZIONE DI UN PLUG-IN 3-WAY PCA PER IMAGE J PER L'ELABORAZIONE DI DATI MULTIVARIATI: APPLICAZIONE ALL'ANALISI MORFOLOGICA DEI NEURONI

Relatori

Prof.ssa Arti Ahluwalia

Ing. Lucia Billeci

Candidata

Chiara Magliaro

Anno Accademico 2010/2011

“Ogni ramo della scienza sembra che ci voglia dimostrare che il mondo si regge su entità sottilissime, come gli impulsi dei neuroni [..] Per l’informatica, è il software che agisce sulle macchine, che esistono in funzione del Software [..] Le macchine di ferro esistono, ma obbediscono ai bit senza peso”

(I.Calvino)

Indice

Introduzione…….1

Capitolo 1……..6

Strumenti Software per l'analisi morfologica di cellule neuronali

1.1 Neurolucida…….7

1.2 Syn-D……...9

1.3 Image J……..13

1.3.1 Neuron J.....14

1.3.2 Neurite Tracer……..17

1.3.3 Neurphology J……..18

1.3.4 NeuronMetrics…….21

1.3.5 Sholl Analysis……...25

1.3.6 FracLac…….28

1.3.7 Neuron_Morpho……...30

1.3.8 Cell Counter…….32

1.4 NeuronStudio……34

1.5 MetaMorph……...36

1.6 TREES toolbox…….39

Capitolo 2……...42

Approccio neuroscientifico allo studio delle alterazioni della morfologia neuronale associate a diversi tipi di patologie

2.1 Disordini dello spettro autistico……..43
2.2 Disordini neurodegenerativi……49
2.2.1 Morbo di Alzheimer……..49
2.2.2 Malattia idiopatica di Parkinson……54
2.2.3 Malattia di Huntington……58
2.3 Alterazioni morfologiche dei neuroni in altre patologie……61

Capitolo 3……...64

Tecniche software

3.1 Ne.Mo……65
3.1.1 Image Processing……..69
3.1.2 Neurons count……...74
3.1.3 Analisi morfologica……..75
3.1.4 Analisi topologica……79
3.1.5 Creazione del datamatrix globale……...81
3.1.6 Plot dei dati morfologici e topologici……..83
3.1.7 Creazione del datamatrix per 3-way PCA.84
3.1.8 3-way PCA……86
3.2 Plug-in Java per la 3-way PCA……...90
3.2.1 Il linguaggio di programmazione Java…..90
3.2.2 Strategie di implementazione del plug-in..93

Capitolo 4……103

Presentazione dei risultati

4.1 Analisi multivariata su neuroni…….104

4.1.1 Costruzione del datamatrix……...104

4.1.2 Analisi dei grafici……...105

4.1.2.1 Cellule wild-type……..105

4.1.2.2 Cellule wild-type e di modello patologico……113

4.1.2.3 Cellule provenienti da claustro..121

4.2 Analisi multivariata su test di tossicità….128

Conclusioni…….137

Appendice A……142

Appendice B……177

Bibliografia…….179

Ringraziamenti……...188

Introduzione

Lo scopo principale di questo lavoro di tesi è stato l'implementazione di un plug-in, in linguaggio di programmazione Java, che effettua la 3-way PCA, tecnica statistica per l'elaborazione di dati multivariati.
Il secondo obiettivo è stato quello di analizzare, tramite questa tecnica, diverse tipologie di dati, in particolare ottenuti dall'analisi morfologica di neuroni, effettuata tramite un tool *open-source*, *Ne.Mo.*, implementato in parte durante il lavoro di tesi specialistica dell'Ing. Lucia Billeci, in parte durante il mio lavoro di tesi triennale.
La 3-way PCA deriva dalla PCA, una tecnica molto utilizzata nei laboratori di biologia e medicina, perché permette l'analisi di dati che provengono dall'osservazione di un certo numero di variabili su un certo numero di soggetti od oggetti. La differenza sostanziale tra le due tecniche è che mentre la PCA viene applicata su una matrice di dati bidimensionale, la 3-way PCA è pensata per lo studio di dati raccolti all'interno di una matrice tridimensionale. Ciò è particolarmente utile in molti studi di ricerca biomedica: spesso, infatti, le variabili sono misurate in un certo numero di condizioni, per cui è necessario aggiungere una terza dimensione alla matrice di dati. La caratteristica fondamentale della 3-way PCA rispetto alla PCA è la capacità di analizzare le correlazioni tra le variabili sulle tre dimensioni: infatti, un datamatrix tridimensionale potrebbe essere analizzato anche applicando la PCA alle matrici bidimensionali che lo compongono, ma in questo modo si potrebbe giungere a conclusioni incomplete, se non addirittura errate. La 3-way PCA, invece, proprio

per la sua capacità di correlazione delle variabili in tre dimensioni, permette all'utente di giungere a conclusioni complete ed esaustive.

Sono stati proposti negli anni diversi modelli di 3-way PCA. In questo lavoro di tesi si è implementato il modello Tucker3, che risulta essere quello più generale.

La scelta del linguaggio di programmazione Java per l'implementazione del plug-in è stata dettata dal fatto che questo, così strutturato, può essere condiviso con la sempre più vasta comunità di Image J.

Image J è un software per l'elaborazione digitale delle immagini, scritto in linguaggio di programmazione Java, e sviluppato dal National Institute of Health (NIH) degli Stati Uniti. E' *open-source*, ed è disponibile per i maggiori sistemi operativi, Microsoft, Mac OS X e Linux. Inoltre, è progettato con un'*open-architecture*, quindi permette ad ogni utente di caricare plug-in che effettuano molteplici elaborazioni su immagini biomediche. L'utente di Image J, dunque, non è solo un fruitore del software: può, infatti, modificare i plug-in già esistenti per l'esecuzione di task desiderati, ma anche condividere con la comunità gli algoritmi da lui implementati. E' per questo che, negli ultimi anni, Image J si è molto sviluppato, arricchendosi di tool che effettuano le più svariate operazioni.

In particolare, il software è provvisto di numerosi plug-in per l'estrazioni di parametri morfologici da cellule neuronali, non però è presente un algoritmo che permetta di interpretare attraverso la 3-way PCA i risultati estratti dalle elaborazioni: da qui l'idea di implementare la 3-way PCA come plug-in per Image J.

Sempre nell'ambito dell'analisi morfologica, esistono numerosi altri software, *open-source* o con licenza, che estraggono informazioni quantitative da immagini ottenute al microscopio che rappresentano cellule neuronali.

L'analisi delle caratteristiche morfologiche neuronali è particolarmente significativa se si analizza lo stato dell'arte riguardante le patologie e, più in generale, le alterazioni dello stato fisiologico che portano a modificazioni della struttura neuronale.

Infatti, gli studi, condotti su tessuti *post-mortem* o *in vivo* tramite le nuove tecniche di *imaging* non invasivo, hanno dimostrato alterazioni sia macroscopiche, come alterazioni del volume cerebrale, che microscopiche, riguardanti, quindi, la struttura della singola cellula, dall'area del soma alla lunghezza dei neuriti. In particolare, numerosi studi confermano la presenza di alterazione della morfologia neuronale negli individui affetti da disordini neurologici, quali ritardi mentali, malattie neurodegenerative o disordini.

L'implementazione di software per la caratterizzazione morfologica e la conta di neuroni *in vitro* durante le prime fasi di sviluppo del cervello è dunque di fondamentale importanza per ottenere misure che siano ripetibili e accurate in tempi ragionevolmente brevi.

I parametri comunemente usati per descrivere la morfologia di neuroni in coltura sono la lunghezza totale dei neuriti, il numero di neuriti primari, il numero di ramificazioni e l'area della regione occupata dall'intero corpo cellulare.

Non tutti i software reperibili in rete, però, conducono analisi complete ed esaustive: ci sono, infatti, molti tool specializzati solo nel pre-processing dell'immagine o solo nell'analisi di una particolare caratteristica della morfologia neurale, ad esempio il calcolo dell'area del soma piuttosto che il *tracing* dei neuriti. La forza di Ne.Mo. rispetto agli altri tool disponibili in rete è che permette all'utente di effettuare con un unico programma tutte le operazioni necessarie per l'estrazione dei parametri morfologici e per la conta neuronale, organizzando automaticamente questi dati in una matrice e visualizzando attraverso dei grafici i risultati ottenuti. Inoltre, rispetto a tutti gli altri software valutati, Ne.Mo. permette di analizzare i dati anche con tecniche di analisi multivariata, tramite la 3-way PCA.

Il presente lavoro di tesi si è articolato in due fasi. Inizialmente ci si è occupati del *debug* del tool Ne.Mo., che è stato implementato, in ambiente Matlab, in parte durante il lavoro di tesi specialistica dell'Ing. Lucia Billeci e in parte durante il mio lavoro di tesi triennale. Inoltre, si sono aggiunti Help in linea e finestre di dialogo con l'utente, in modo da rendere il software il più *user-friendly* possibile:

ciò è di fondamentale importanza, essendo esso destinato a laboratori biologici, in cui non è garantita la presenza di operatori con buone conoscenze informatiche. Una volta controllata la corretta funzionalità delle varie interfacce grafiche, con i relativi editor che lo compongono, si è proceduto con la compilazione del codice, in modo da ottenere un file eseguibile. In questo modo, Ne.Mo. è utilizzabile anche su computer sprovvisto della licenza MathWorks. Infine, si è proceduto con la registrazione il software, con i soli diritti morali e non di utilizzazione economica, con il nome NEMO: NEuron MOrphological Analysis Tool 1.0

La seconda fase del lavoro è quella più consistente e costituisce la parte centrale della tesi. Si è infatti implementato, come già ribadito precedentemente, un plug-in in linguaggio di programmazione Java che permette di elaborare dati multivariati.

Gli algoritmi che effettuano la 3-way PCA erano già stati implementati, da Leardi e collaboratori, in ambiente Matlab. E' stato quindi necessario effettuare un'operazione di "traduzione" da un linguaggio di programmazione a un altro, tenendo conto delle differenze sostanziali tra i due. Matlab è, infatti, un ambiente di calcolo numerico, che manipola fondamentalmente matrici (Matlab è l'acronimo di MAtrix LABoratory): per questo sono a disposizione del programmatore le più comuni operazioni matriciali, oltre ad essere disponibili tool a supporto dei più disparati campi di studio; al contrario, Java è un linguaggio di programmazione orientato agli oggetti, e fornisce la possibilità al programmatore di implementare metodi per la creazione di costrutti di cui invece Matlab dispone nativamente: in Java, infatti, sono disponibili le sole operazioni algebriche.

Per testare l'effettiva validità del plug-in implementato, si sono analizzati sia con lo script in Java che con quello in Matlab quattro matrici di dati, verificando che i risultati ottenuti fossero effettivamente gli stessi.

Le prime tre matrici di dati raccoglievano al loro interno caratteristiche morfologiche di neuroni in coltura. Le prime due

matrici sono state ottenute analizzando immagini di cellule di Purkinje estratte da topi wild-type e da modelli di autismo e fotografate in diversi giorni di coltura, mentre la terza è stata ottenuta fotografando neuroni fissati provenienti da slice a diverse profondità di claustro, sottile lamina di sostanza grigia situata nella corteccia dell'insula.

Per sottolineare, infine, come questo tipo di analisi multivariata si possa utilizzare nei più disparati campi di ricerca, si sono analizzati datamatrix non solo riguardanti l'analisi morfologica neuronale, ma in particolare una matrice contenente dati relativi alla la vitalità di tre diverse colture cellulari (epatociti –C3A-, cellule endoteliali -HUVEc- e cellule epiteliali intestinali –CACO-2-) esposte a diverse concentrazioni di nanoparticelle tossiche.

Capitolo 1

Strumenti software per l'analisi morfologica di cellule neuronali

La forma, la struttura e la connettività neuronale sono aspetti fondamentali per la comprensione della funzione nervosa. Numerosi studi confermano la presenza di alterazione della morfologia neuronale negli individui affetti da disordini neurologici, quali ritardi mentali, malattie neurodegenerative o disordini psichiatrici (**Lin** e **Koleske**, 2010). Studi in vitro e modelli animali sono largamente utilizzati per condurre questo particolare tipo di ricerca sui meccanismi molecolari alla base delle malattie al cervello (**Groffen et al.**, 2010; **Jockusch et al.**, 2007; **Kawabe et al.**, 2010; **Priller et al**., 2007).

Poter quantificare in maniera consistente i differenti aspetti della morfologia neuronale è di vitale importanza per ottenere informazioni sui *pathway* alla base dei meccanismi patogeni. Fino a pochi anni fa, l'analisi morfologica su immagini ottenute al microscopio era effettuata manualmente. Ciò risultava essere molto laborioso, soprattutto quando erano richieste misure particolari, come la caratterizzazione dell'albero dendritico, e molto dispendioso, in termini di tempo e risorse umane. A ciò si aggiungeva la diminuzione del livello di riproducibilità della misura, dovuto alla differenza nel

valore delle grandezze quando venivano effettuate più misurazioni, sia da parte di due operatori diversi, sia dello stesso operatore in tempi diversi. Per questo motivo, negli ultimi venti anni sono stati sviluppati vari tool in grado di analizzare le caratteristiche morfologiche delle cellule neuronali attraverso l'*image processing*. I parametri comunemente usati per descrivere la morfologia di neuroni in coltura sono la lunghezza totale dei neuriti, il numero di neuriti primari, il numero di ramificazioni e l'area della regione occupata dall'intero corpo cellulare. A questi si aggiungono parametri secondari, come la polarità, misura dell'estensione assonale relativamente a quella dendritica. Esistono poi grandezze specifiche per i vari tipi neuronali, come ad esempio la curvatura dei neuriti.

In questo capitolo si descriveranno tool che effettuano l'analisi morfologica a partire da immagini di neuroni ottenute al microscopio o tramite MRI. Ce ne sono diversi attualmente disponibili in rete: si è operata una scelta, descrivendo solo i software maggiormente utilizzati negli articoli più recenti. Attenzione maggiore è stata data ad ImageJ, software open - source distribuito gratuitamente, che da qualche anno è divenuto uno dei programmi più utilizzati nei laboratori mondiali. Dei tool *open-source*, di cui è spesso disponibile anche il codice sorgente, si analizzeranno più nel dettaglio gli algoritmi che permettono il *processing* e l'analisi; ciò non è possibile per quelli che richiedono l'utilizzo di una licenza. Le notizie riportate sono state reperite sia da paper che dai manuali e brochure tecniche. Non tutti i software effettuano lo stesso tipo di analisi: si descriveranno tool per il solo riconoscimento dell'albero dendritico, ma anche di altri che effettuano solo image processing e analisi morfometrica. Si cercherà di operare un confronto tra i vari tool, sottolineandone analogie e differenze.

1.1 Neurolucida [1][2][3]

Il sistema più comunemente usato per il *brain mapping*, la ricostruzione neuronale e l'analisi morfometrica è il package commerciale Neurolucida (MicroBrightField, Williston, VT), scritto

in linguaggio C. La prima versione risale oramai a 20 anni fa, ma ha continuato ad evolversi nel tempo, diventando il *gold standard* internazionale nel campo della neuroscienza cellulare. L'interfaccia user-friendly (Fig. 1) permette di ottenere risultati in poco tempo, acquisendo dati e ricostruendo l'intera estensione neuronale delle regioni cerebrali volute. Inoltre, permette analisi morfologiche quantitative, presentate sia attraverso grafici, sia su foglio elettronico in formato compatibile con Excel. Infine, esegue il processing di immagini acquisite in vari modi, dal microscopio confocale alla MRI.

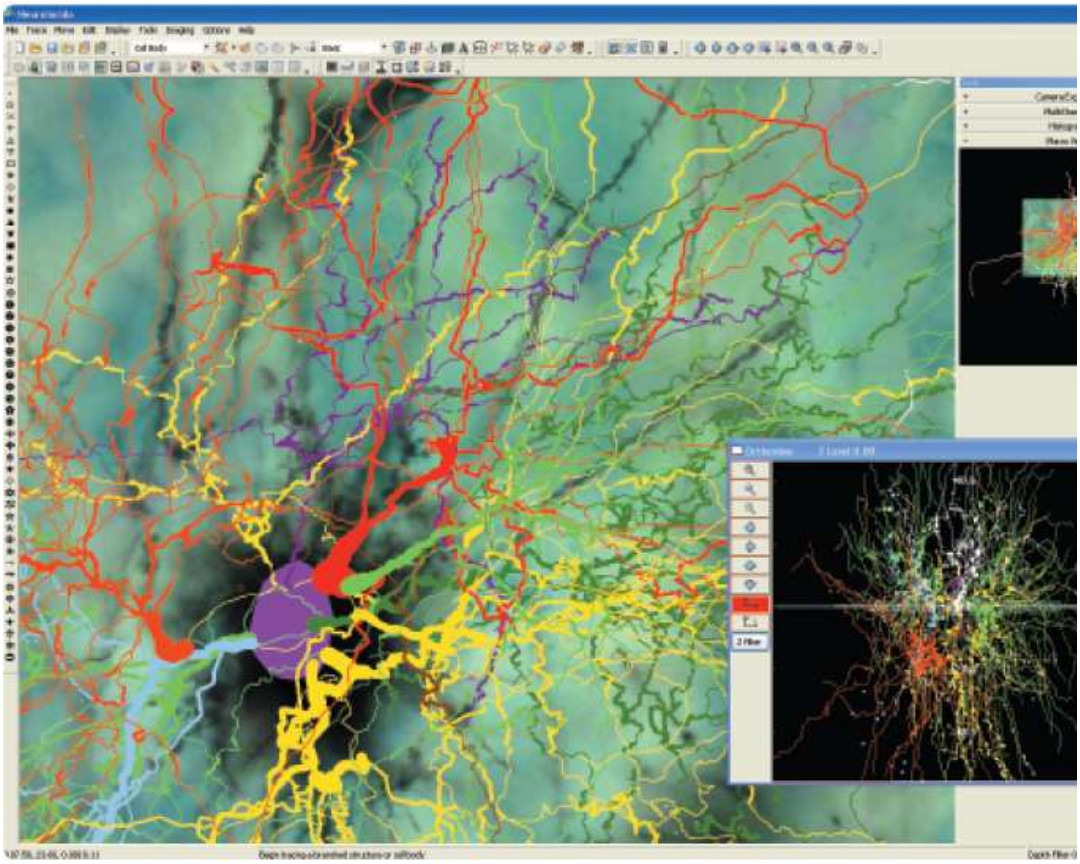


Fig. 1 Interfaccia Grafica di Neurolucida

Neurolucida permette di:

- Tracciare neuroni, identificando bottoni sinaptici e spine dendritiche: ciò permette anche una loro classificazione. Tale operazione è fatta automaticamente attraverso il modulo *Autoneuron*;
- Eseguire un *mapping* anatomico, attraverso vari tipi di *marker* per l'identificazione di particolari tipi di strutture nel cervello;
- Effettuare l'*Image processing*, che permette di correggere le anomalie del background delle immagini, dovute all'irregolarità di

illuminazione, di utilizzare filtri che permettono l'ottenimento dei contorni degli oggetti, si riconoscerli automaticamente e di migliorarne del contrasto;

- Eseguire l'analisi morfologica dell'albero dendritico, ricavando parametri quali la lunghezza dei dendriti, il loro numero, l'angolo di ramificazione o la distribuzione delle spine dendritiche o dei bottoni sinaptici. Effettua inoltra l'analisi di Sholl ed analisi di tipo statistico per l'ottenimento del dendrogramma;
- Eseguire l'analisi su vari tipi cellulari, non necessariamente neuroni, per ottenere informazioni sulla forma o sulla grandezza;
- Effettuare la conta cellulare, basato su tecniche stereologiche *design-based*, cioè metodi per il riconoscimento di forme non basati sulle proprietà geometriche degli oggetti studiati (tecniche *model-based*, sorgenti di errori sistematici), ma i cui schemi sono definiti *a priori*.
- Visualizzare le strutture in tre dimensioni;

La precisione nella ricostruzione dendritica e nel *mapping* anatomico è di 0.5 μm.

Oltre al software, non *open-source*, e ai vari moduli estensivi dello stesso (come *Autoneuron*, citato precedentemente, o *Image Stack Module*, che opera su *stack* di immagini ottenute tramite microscopia confocale), la MicroBrightField offre una fotocamera digitale o una videocamera a cattura di frame, un sistema per la messa a fuoco automatica, una workstation e hardware specializzati per determinati compiti, ad esempio filtri, sistema di illuminazione ed encoder per stabilire la coordinata z.

Neurolucida è un *package* completo, ma può essere utilizzato solo sul sistema operativo Microsoft Windows. Altro fattore che fa desistere dall'utilizzo in molti laboratori interessati alla ricostruzione digitale della morfologia neuronale è il costo considerevole.

1.2 Syn-D [4]

Syn-D, anagramma di **Syn**apse **D**etection, è un programma che effettua l'analisi automatica della morfologia neuronale. Ha

un'interfaccia grafica intuitiva (Fig. 2), che può essere quindi utilizzata anche da operatori con poche conoscenze informatiche. Il programma, implementato in ambiente Matlab e utilizzabile indipendentemente in Windows e MacOS, è distribuito gratuitamente.

Syn-D, data un'immagine ottenuta fotografando neuroni fluorescenti in coltura, riconosce in modo automatico soma, dendriti e sinapsi, e quantifica parametri come la lunghezza dei dendriti o il numero delle sinapsi, effettua l'analisi di Sholl per ottenere informazioni sulla struttura dell'albero dendritico, localizza le sinapsi, riporta l'area del soma. Infine, quantifica l'intensità della fluorescenza della sinapsi, riporta numero, dimensione, posizione e densità di molti organelli cellulari, ad esempio lisosomi, e può essere usato per effettuare la conta dei neuroni in *slice* di cervello, fissate o non.

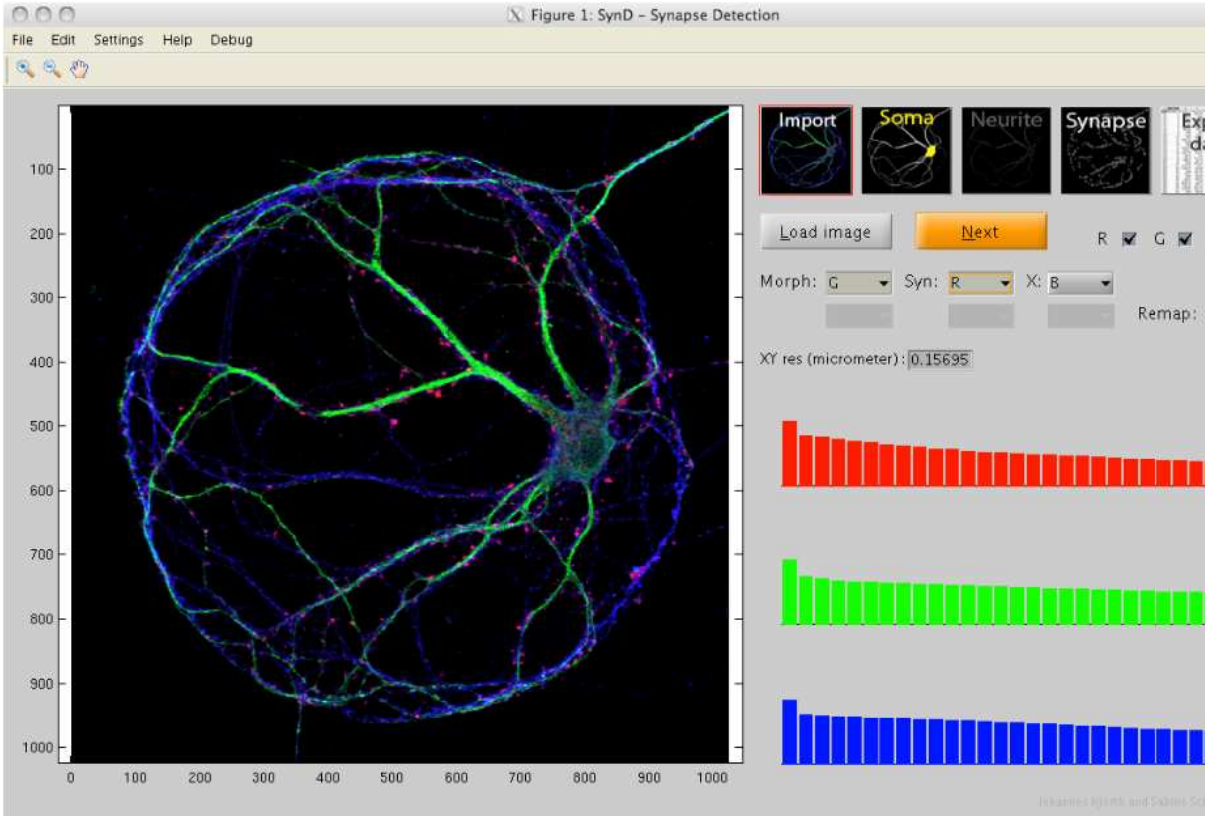


Fig.2 Interfaccia Grafica di Syn-D

La procedura di *detection* e *processing* è schematizzabile in cinque step:

- Caricamento dell'immagine

L'utente seleziona un'immagine a colori e specifica in quale dei tre canali (R, G o B) è contenuta l'informazione morfologica, tenendo

conto della colorazione della coltura. A seconda del formato dell'immagine, è necessario specificare manualmente o meno la dimensione di un singolo pixel.

- Riconoscimento del soma

Utilizzando un filtro passa-basso, in particolare un filtro di Wiener adattativo, si riduce il rumore di background dell'immagine, assumendo che questo sia di tipo gaussiano. Con un'operazione di sogliatura, si ottiene un'immagine binaria, a sfondo nero con il neurone ben in evidenza in bianco. Con un'operazione morfologica di opening, utilizzando un *kernel* di opportune dimensioni, si riesce a separare il soma dai neuriti.

- Riconoscimento dei neuriti

I neuriti sono riconoscibili visivamente perché circondano il soma a creare una struttura ramificata, più luminosa rispetto allo sfondo scuro. Per l'identificazione di tale struttura, si utilizzano dei filtri, implementati attraverso opportuni comandi Matlab, basati sulle derivate di ordine superiore della gaussiana G, applicate all'immagine f:

$$f_{ij} = -(f * G_{ij})$$

con

$$G_{ij} = \frac{\partial^2 G}{\partial_i \partial_j}$$

in cui il simbolo * indica la convoluzione spaziale, e gli indici i e j le direzioni x e y rispettivamente. Si utilizzano, quindi, gli autovalori e gli autovettori della matrice Hessiana H,

$$H = \begin{pmatrix} f_{xx} & f_{xy} \\ f_{yx} & f_{yy} \end{pmatrix}$$

per calcolare un'opportuna funzione costo, che, dato il pixel P_{ij}, tiene conto in maniera ponderata sia della similarità tra le intensità di tale pixel con gli adiacenti, sia della similarità tra la direzione del vettore che collega il P_{ij} a quello precedentemente considerato e la direzione media della ramificazione intorno al P_{ij}. L'algoritmo parte da un punto, detto "seme", e calcola la funzione costo per tutti i pixel nell'intorno. I pixel la cui intensità è al di sotto della soglia stabilita

dalla funzione costo sono considerati appartenenti alla struttura dei neuriti. Questa procedura è iterata fino a che nell'immagine non ci saranno più pixel che soddisfano questa condizione. Per quanto riguarda i neuriti più sottili, diversamente da Meijering che propone un riconoscimento di tipo manuale, Syn-D, partendo dal soma, include pixel alla struttura neurale fino a che la funzione costo è minore di una soglia, imposta uguale a 0.9. Ciò permette di aggiungere alla struttura i neuriti più sottili, ma non rumore o regioni spurie. A causa delle successive elaborazioni sull'immagine, è possibile che i neuriti non siano tutti collegati al soma a formare l'intera arborizzazione dendritica: Syn-D automaticamente connette i vari rami.

- Riconoscimento della sinapsi

Le sinapsi si presentano come delle regioni più chiare all'interno dell'immagine. Tramite un'operazione di sogliatura, Syn-D considera appartenenti alla sinapsi i pixel il cui valore di intensità è maggiore della somma tra la media delle intensità dei pixel del canale delle sinapsi e la deviazione standard del pixel considerato. Considerando però che fisiologicamente le sinapsi si sviluppano in prossimità dei dendriti, verranno escluse dal set di strutture rilevate quelle troppo distanti dai questi, come anche verranno escluse regioni più piccole di 0.35 μm^2. Nel caso in cui ci siano cluster di sinapsi, è necessario identificare le singole strutture. Il programma cerca all'interno dell'immagine regioni sinaptiche caratterizzate dall'avere un'unica intensità locale massima. Queste vengono usate per costruire il cosiddetto *kernel* sinaptico, che permette di identificare le sinapsi ad una ad una, operando una deconvoluzione tra l'immagine e l'elemento stesso.

- Analisi dei dati

In quest'ultimo step sono calcolati vari parametri morfologici.

Per il calcolo della lunghezza dei dendriti, è necessario ottenere lo scheletro dell'immagine, cioè la riduzione della struttura neuronale a linee di spessore un pixel.

Il numero delle sinapsi presenti nell'immagine è ottenuto tramite la deconvoluzione descritta nello step precedente.

Per quantificare l'arborizzazioni denditrica, Syn-D implementa in metodo di Sholl, costruendo automaticamente dei cerchi concentrici di raggio sempre crescente, centrati sul soma. Il cerchio più piccolo ha raggio uguale al raggio maggiore del soma, approssimato a un'ellisse.
Syn-D inoltre riporta le coordinate dei centri di ogni sinapsi, e costruisce un istogramma in cui riporta il numero di sinapsi tra due cerchi di Sholl consecutivi. Si riporta anche una misura dell'intensità della fluorescenza all'interno delle sinapsi individuate, anche considerando i tre canali, rosso, blu e verde separatamente, comparandone infine le misure medie.
Infine, il programma calcola area e lunghezza del raggio minimo e massimo del soma.
Syn-D è un tool molto potente per l'analisi della morfologia neuronale: è efficiente dal punto di vista del tempo necessario ad ottenere i dati, ed essendo totalmente automatizzato non è soggetto a bias e ad errori umani.

1.3 Image J [5]

Image J (Fig. 3) è un software per l'elaborazione digitale delle immagini, scritto in linguaggio di programmazione Java, e sviluppato dal *National Institute of Health* (NIH) degli Stati Uniti. E' open source, ed è disponibile per i tre maggiori sistemi operativi, Windows, Mac OS X e Linux.
Può elaborare immagini di vario tipo, e salvate in diversi formati, tra cui TIFF, GIF, JPEG, BMP, DICOM e FITS.
E' un software progettato con una *open-architecture*, permette, quindi, ad ogni utente di caricare tool che effettuano particolari operazioni. E' per questo che, negli ultimi anni, ImageJ si è molto sviluppato, arricchendosi di plug-in che eseguono le più svariate operazioni, dal semplice processing delle immagini, quali la sogliatura o il miglioramento del contrato, a operazioni complesse, quali la ricostruzione in 3D a partire da *stack* di immagini.
Come sottolineato da **R. Stallman** (1986), un utente di ImageJ ha la libertà di:

- utilizzare i programmi scaricabili sul sito di ImageJ per qualsiasi tipo di studio;
- studiare gli algoritmi che compongono il programma, modificandoli per eseguire il task desiderato;
- condividere le modifiche con gli altri utenti;

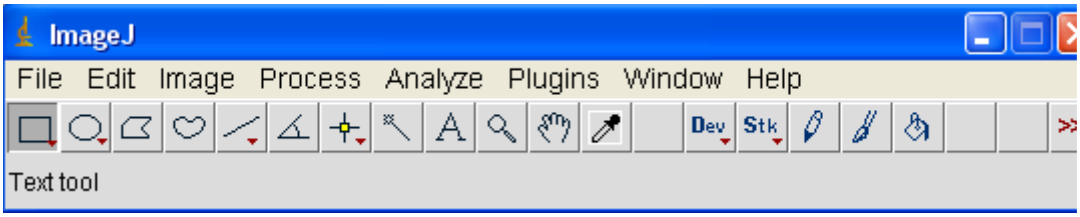


Fig. 3 Interfaccia di Image J

In Image J, è possibile trovare vari tool che permettono di effettuare la ricostruzione neuronale e\o l'analisi morfologica di neuroni:

- Neuron J
- NeuriteTracer
- Neurphology J
- NeuronMetrics
- Sholl Analysis
- FracLac
- Neuron_Morpho
- Cell counter

che si esamineranno nel dettaglio.

1.3.1 NeuronJ [6][7]

NeuronJ (Fig. 4) è un plug-in di Image J, quindi è scritto in linguaggio di programmazione Java ed è distribuito gratuitamente. E' un'applicazione pensata per immagini in 2D in scala di grigio o a colori.

L'idea alla base del software sta nella necessità di avere uno strumento che, in modo riproducibile e accurato, permettesse di individuare i neuriti in un'immagine rappresentanti neuroni in fluorescenza in coltura. Poiché le procedure di sogliatura e di scheletrizzazione, usate in molti dei software reperibili in rete, non

sono affidabili nel ricostruire la struttura neuronale (dipendono molto, infatti, dalla qualità dell'immagine di partenza), si sono sviluppati metodi alternativi, documentati in letteratura, per il riconoscimento dei neuriti.

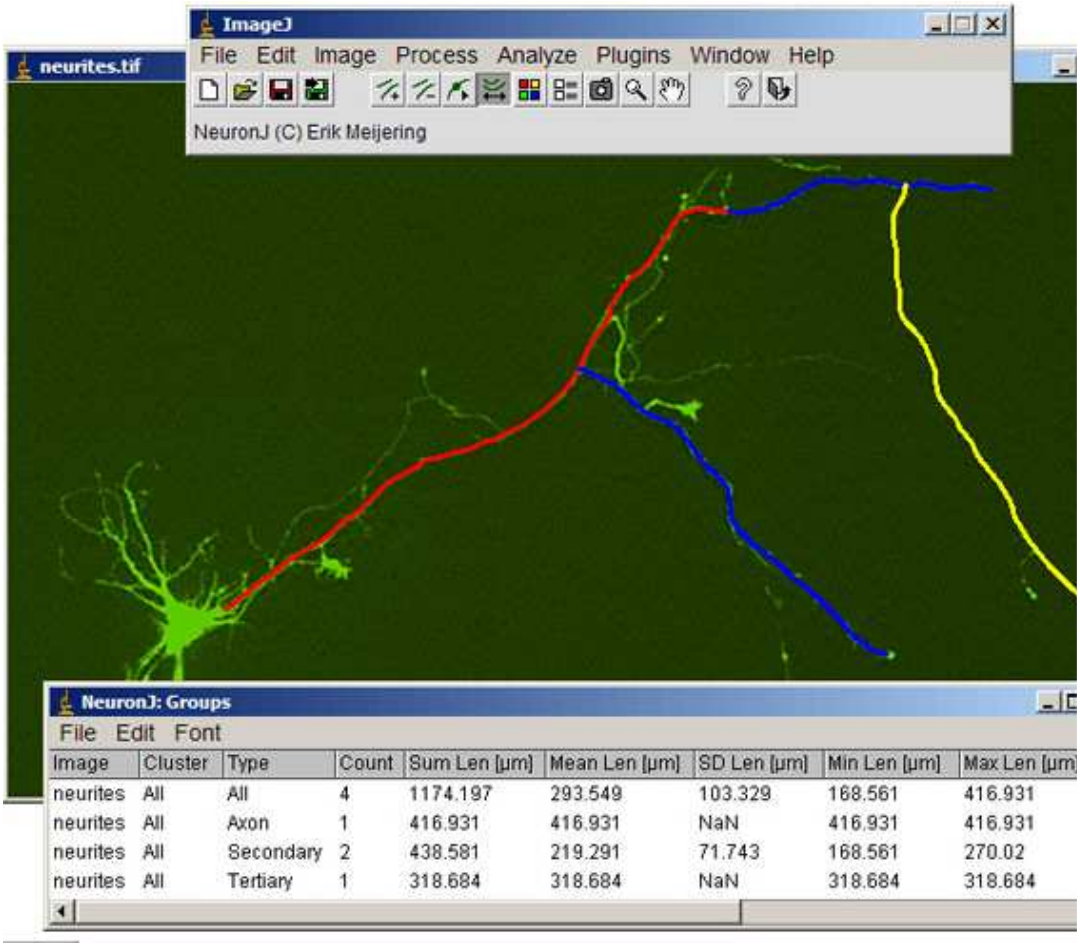


Fig. 4 Interfaccia grafica Neuron J

Il riconoscimento dei neuriti in Neuron J consiste in due fasi:

- Detection-phase, in cui ad ogni pixel è assegnato un valore indicante la probabilità che sia appartenente ad un neurite. E' uno step completamente automatico.
- Tracing-phase, in cui i pixel consecutivi che con probabilità maggiore rispetto ad altri costituiscono il neurite sono collegati a formare la struttura finale. In questo step, è necessario l'intervento dell'utente.

  Nell'implementazione dell'algoritmo, si utilizza un'operazione di convoluzione tra l'immagine e la derivata seconda di un *kernel* gaussiano. Matematicamente, quindi, data un'immagine f e una gaussiana normalizzata G, l'operazione eseguita sarà:

$$f_{ij}(x) = (f * G_{ij})(x)$$

con

$$G_{ij}(x) := \frac{\delta^2 G(x)}{\delta i \delta j}$$

E' lo stesso procedimento utilizzato in un altro software, Syn-D, di cui si parlerà in seguito. La differenza tra i due software è nella matrice da cui si ottengono autovalori e auto vettori, che per NeuronJ non è la semplice Hessiana:

$$H = \begin{pmatrix} f_{xx} + \alpha * f_{yy} & (1-\alpha) * f_{xy} \\ (1-\alpha) * f_{xy} & f_{yy} + \alpha * f_{yy} \end{pmatrix}$$

in cui α è un parametro che dà risultati ottimali se fissato a -1/3.
Basandosi sul valore degli autovalori, l'algoritmo assegna ad ogni pixel dell'immagine un indice della probabilità che questo sia appartenente al neurite, in accordo con la seguente formula:

$$\rho(x) = \frac{\lambda(x)}{\lambda_{\min}} se \lambda(x) < 0$$

$$\rho(x) = 0 se \lambda(x) \geq 0$$

in cui λ è l'autovalore del pixel in posizione x e λ_{min} è l'autovalore minore tra tutti i pixel. Inoltre, l'algoritmo per ogni pixel calcola il parametro v, indice dell'orientamento della ramificazione del neurite, come rapporto tra l'autovettore corrispondente al più piccolo autovalore, preso in valore assoluto.
La funzione costo utilizzata in Neuron J per decidere se il pixel y è connesso al pixel x è:

$$C(x, y) = \gamma * C_{\lambda}(y) + (1-\gamma) * C_{v}(x, y)$$

con

$$C_{\lambda}(y) = 1 - \rho(y)$$

e

$$C_{v}(x, y) = \frac{1}{2} * [\sqrt{1-\phi(x, y)} + \sqrt{1-\phi(y, x)}]$$

con γ parametro, che può assumere valori compresi tra 0 e 1, determinante il peso che i due addendi nel calcolo di C (x, y), e φ(x, y) = |v(x)(x-y)/(||y-x||)|, che rappresenta il vettore-connessione tra i due pixel.

Il metodo implementato in Neuron J per il riconoscimento dei neuriti è basato sull'osservazione delle immagini delle cellule in fluorescenza in colture bidimensionali: queste appaiono come strutture allungate e chiare, al contrario dello sfondo, scuro e affetto da rumore. Rappresentando l'immagine a livelli di grigio, il neurite si mostra come una struttura composta fondamentalmente di linee, che sono ben rivelate con operatori che implementano la derivata seconda.

L'utente seleziona un pixel di partenza appartenente al neurite, e l'algoritmo cerca il percorso ottimale, ossia quello con funzione di costo minima, e tra il punto selezionato e gli otto punti che lo circondano. Il processo è iterato fino a che non è riconosciuto l'intero neurite, e può essere ripetuto per anche per più neuriti. Nel caso in cui l'utente non è pienamente soddisfatto del risultato ottenuto, è possibile riconoscere la struttura anche manualmente. Da questo metodo è possibile ricavare, in maniera semplice, la lunghezza dei neuriti, e il numero di soma presenti nell'immagine. Uno dei limiti dell'algoritmo è che non presenta nessun altro tipo di operazione, come ad esempio una quantificazione della grandezza del soma cellulare.

Il metodo semiautomatico di ricostruzione della struttura neuronale è valido in termini di riproducibilità e tempistiche. E' un plug-in dedicato solo alla ricostruzione delle immagini, non effettua quindi una rigorosa analisi morfologica.

1.3.2 Neurite Tracer [8][9]

NeuriteTracer è un plug-in di ImageJ che esegue il *tracing* neuronale. A differenza di NeuronJ, l'algoritmo minimizza l'intervento dell'utente, pur mantenendone l'accuratezza nell'identificazione della struttura neuronale. Opera su immagini in 2D, ottenute tramite microscopia e rappresentanti neuroni in coltura. L'algoritmo ha bisogno di due immagini distinte: una ottenuta marcando l'intero neurone, e l'altra acquisita con uno specifico *marker* per il soma. Le prime fasi di elaborazione, fatte con strumenti già implementati in ImageJ, sono comune a entrambi i tipi di fotografia. Infatti, le immagini sono dapprima pre-elaborate, per ottimizzare l'uniformità

dell'illuminazione e il contrasto, poi sono rese binarie attraverso un'operazione di sogliatura, e sono ripulite, tramite opportune operazioni morfologiche, da oggetti non rilevanti per l'analisi dell'albero dendritico. Infine, si identificano i pixel presenti nella prima e nella seconda immagine. A questo punto, si effettuano elaborazioni specifiche. Eseguendo una sottrazione tra i pixel bianchi nella prima e nella seconda immagine, si ottiene una figura binaria rappresentante i soli neuriti: scheletrizzandola, si effettua il *tracing* neuronale. Infine, partendo dall'immagine raffigurante solo i soma cellulari, si esegue la conta con uno specifico algoritmo. A questo punto, NeuriteTracer automaticamente crea un file .txt in cui riporta il numero di soma (ma non la dimensione degli stessi) presenti nell'immagine, e la lunghezza dei neuriti.

Il *tool* ricostruisce in maniera fedele e dettagliata la struttura dell'albero dendritico (utilizzando le stesse cellule, elaborate con Neurite Tracer e NeuronJ, si ottengono risultati comparabili [9]) ma non permette il calcolo di tutti i parametri necessari per un'analisi completa della morfologia neuronale.

1.3.3 Neurphology J [10]

Neurphology J (Fig. 5) è un metodo implementato al fine di quantificare la morfologia neuronale in immagini 2D. L'algoritmo cerca di:

- Minimizzare l'intervento dell'utente;
- Rendere l'algoritmo più conveniente rispetto a NeuronJ e NeuriteTracer, già presenti in ImageJ;
- Massimizzare la velocità di esecuzione;
- Massimizzare l'accuratezza dell'esecuzione del *tracing* e delle misure;
- Ottenere un algoritmo affidabile e sicuro.

L'algoritmo consiste di cinque parti, una di miglioramento della qualità dell'immagine, per renderla idonea alle successive elaborazioni, e le altre quattro di quantificazione morfologica:

- Image Processing: costituito da più operazioni che aumentano il rapporto segnale-rumore, in modo da poter calcolare più facilmente la

soglia necessaria a rendere l'immagine binaria: è su questo tipo di immagine, infatti, che si basa l'estrazione dei parametri morfologici. Per migliorare la qualità dell'immagine si effettua il rilevamento contorni, la correzione del background, in modo da renderlo più uniforme, e la selezione dei pixel in base alla loro intensità. A questo punto, il software calcola automaticamente la soglia che permetterà all'immagine a livelli di grigio di diventare immagine binaria, evidenziando la cellula in bianco, rispetto allo sfondo scuro.

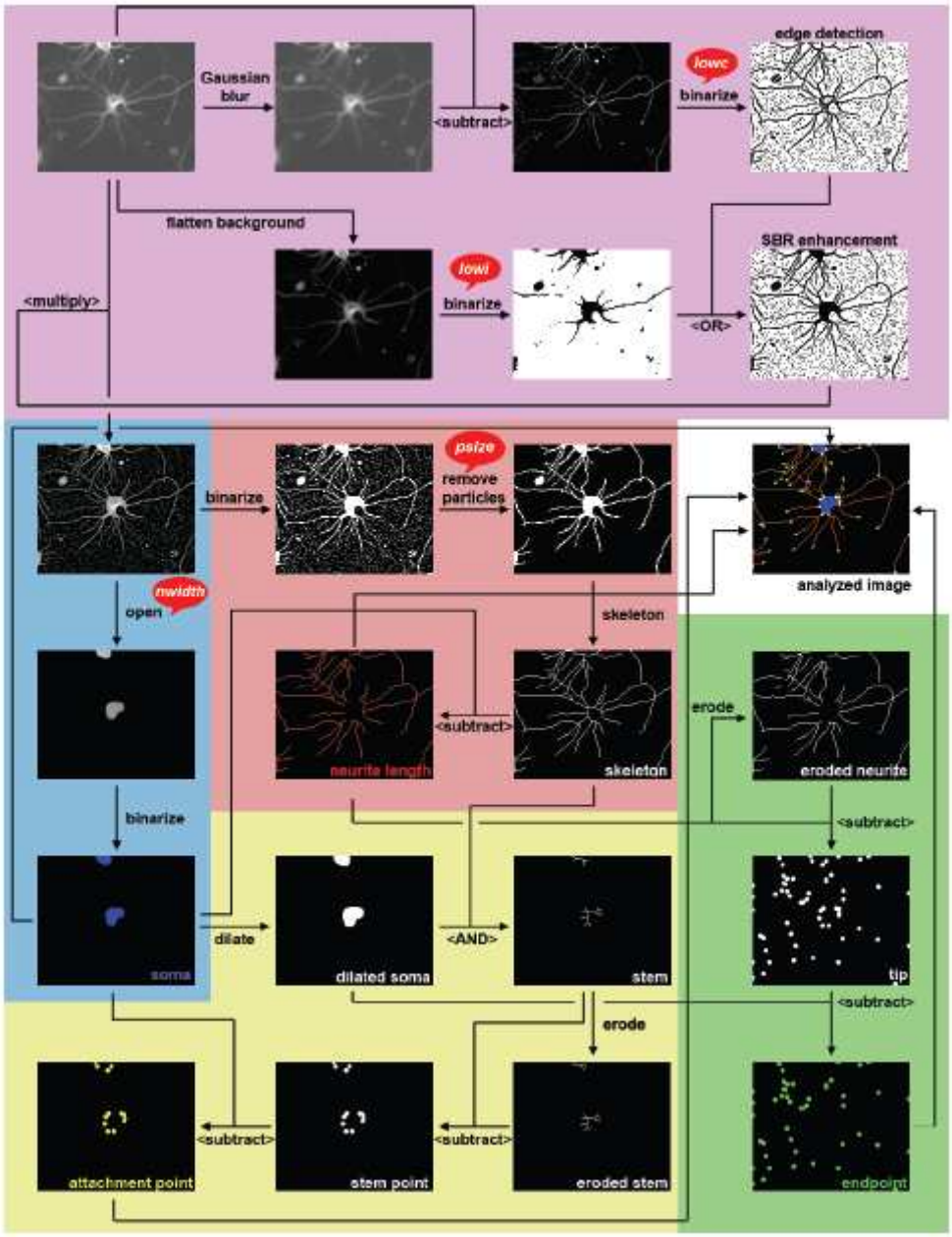


Fig. 5 Schema dei principali step eseguiti da NeurphologyJ

- Estrazione e quantificazione del soma: con un'operazione morfologica di apertura, si isola il soma. In questo modo, si ottiene un'immagine a sfondo nero, con un oggetto bianco di forma approssimabile ad un'ellisse, rappresentante il soma.
- Estrazione e quantificazione della lunghezza dei neuriti: dopo aver eliminato i pixel non interessanti dal punto di vista dell'analisi che si vuole condurre, all'immagine scheletrizzata viene sottratta l'immagine rappresentante il solo soma: il risultato sarà la sola struttura neuritica. La lunghezza totale di dendriti e assoni è quantificata contando il numero di pixel nell'immagine.
- Identificazione dei punti di adesione dei neuriti al soma: si esegue una serie di operazioni che permettono di ottenere un'immagine con dei "*tip*", cioè dei punti che identificano la posizione punto in cui il neurite è connesso al soma. Con un'operazione morfologica di dilatazione sull'immagine che rappresenta il solo soma, si aumentano le dimensioni di quest'ultimo. L'immagine risultante è combinata con lo scheletro del neurone attraverso un AND logico (operazione possibile perché entrambe le immagini sono binarie). A questo punto, si ottiene una struttura formata da linee di spessore un pixel in corrispondenza del soma. Infine, si opera una sottrazione tra l'immagine ed essa stessa, previa erosione: l'utente otterrà la figura richiesta.
- Identificazione dell'estremità dei neuriti: si esegue una serie di operazioni che permettono di ottenere un'immagine con dei "*tip*" che identificano la posizione dell'estremità del neurite più lontana dal soma. Per ottenerla, si esegue dapprima una sottrazione tra l'immagine rappresentante i neuriti, ottenuta negli step precedenti, e la stessa immagine previa erosione. Infine, si sottrae all'immagine ottenuta quella rappresentante i soli neuriti.

  I risultati che si ottengono sono comparabili a quelli di NeuronJ, ma il procedimento è completamente automatico, e, oltre alla lunghezza dei neuriti e alla quantificazione del numero di soma presenti in un'immagine, NeurphologyJ quantifica l'area del soma stesso, e localizza la posizione delle estremità prossimali e distali dei neuriti.

1.3.4 NeuronMetrics [11]

Scritto come modulo di ImageJ, NeuronMetrics è un software per l'analisi quantitativa semi-automatica di immagini in 2D di neuroni in coltura. NeuronMetrics, infatti, esegue operazioni su immagini a livelli di grigio a 8 o 16 bit ad alto contrasto: ciò riduce i tempi di *pre-processing*. Tuttavia, è comunque necessaria una elaborazione dell'immagine, al fine di ottenere lo scheletro degli oggetti in essa rappresentati. L'utente, tramite la finestra di *Setup*, indica la locazione della cartella contenente le immagini da elaborare e la dimensione dei pixel, dopodiché si procede con l'analisi vera e propria. Il tool elabora automaticamente tutte le immagini presenti nella cartella selezionata. I moduli principali del software, interessanti dal punto di vista dell'elaborazione, sono quattro (Fig. 6):

- Neuron & Cell Body: è costituito a sua volta da quattro moduli, che permettono all'utente di "ritagliare" la regione dell'immagine in cui è presente il neurone da elaborare (*Neuron ROI*), di rimuovere eventuale rumore (*Noise ROIs*), di osservare in modo sequenziale le immagini presenti nella cartella selezionata

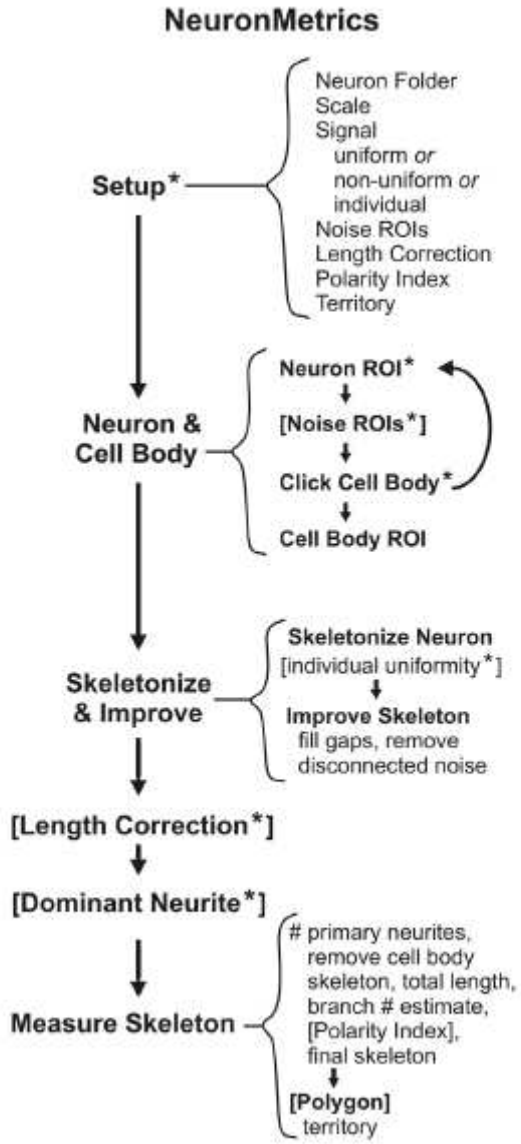


Fig. 6 Schema degli step da effettuare durante l'utilizzo di NeuronMetrics

nel Setup dall'utente, selezionando in ognuna, con un click, l'oggetto interessante dal punto di vista della successiva elaborazione (*Click Cell Body*). E' necessaria una rappresentazione approssimativa del perimetro del soma cellulare, in modo da escludere i pixel che lo costituiscono dal conteggio della lunghezza totale dei neuriti. NeuronMetrics utilizza un algoritmo di sottrazione, in modo da ottenere un'immagine in cui la cellula è descritta da pixel con lo stesso valore di grigio, seguito da un filtro di Sobel per il riconoscimento dei contorni del neurone. Con lo strumento "*wand*" di ImageJ, ottengo il perimetro della cellula.

- Skeletonize & Improve, con cui si ottiene l'immagine rappresentante lo scheletro del neurone. Il processo è completamente automatico, per tutte le immagini contenute nella cartella selezionata nel Setup. L'immagine ha bisogno di un pre-*processing*, eseguito dall'utente in maniera empirica. Il risultato è utilizzato per creare due tipi di immagine: quella binaria, tramite un'operazione di sogliatura, che mostra in modo chiaro i neuriti, il soma e i punti di biforcazione, e quella laplaciana, che mette in evidenza anche i neuriti più piccoli. Queste due immagini sono fuse insieme con un opportuno algoritmo di *merge*, e il risultato è scheletrizzato. Si tratta, in realtà, di una scheletrizzazione preliminare: il neurone è rappresentato da linee di spessore un pixel, ma contiene spesso ancora del rumore, dovuto per la maggior parte a oggetti non interessanti dal punto di vista dell'analisi morfologica. E' necessario, quindi, elaborare ancora l'immagine, eliminando il rumore e il soma dallo scheletro, e connettendo i neuriti che a causa del processing o dell'acquisizione risultano distaccati.

  L'algoritmo di connessione utilizza tre grandezze empiricamente determinate, dette rispettivamente *gap distance*, *extend distance* e *maximum deviation*. Dato un pixel p all'estremità dalle curva c_1 e un pixel q all'estremità della curva c_2, automaticamente viene disegnata una linea retta di spessore un pixel che collega p a q se la distanza tra i due è minore di un certo valore, detto *gap distance*. La linea retta è estesa oltre i punti p e q di una lunghezza pari all' *estend distance*, le cui estremità sono p' e q'. Se la distanza minima tra p' e c_1 e tra q' e c_2 è minore della *maximum deviation*, c_1 e c_2 sono considerati appartenenti o stesso oggetto, e sono quindi connessi. E' previsto un "close test" per verificare che effettivamente si siano connessi tutti i neuriti appartenenti alla stessa struttura neuronale.

  Applicato l'algoritmo di connessione, tutto ciò che non è connesso allo scheletro è eliminato perché considerato rumore. A questo punto, si ottiene lo scheletro idoneo alle successive elaborazioni.

- Measure Skeleton: calcola parametrici morfologici, quali lunghezza totale dei neuriti, numeri di ramificazioni, numero di neuriti primari e area della superficie occupata dalla cellula.

  Per quanto riguarda la lunghezza dell'albero neuritico, questa è calcolata partendo dagli apici di ogni neurite primario e spostandosi lungo la struttura cellulare, assegnando valore unitario agli step compiuti in orizzontale o verticale, e valore $\sqrt{2}$ agli step in diagonale. La lunghezza totale sarà la somma di tutti gli step effettuati. Poiché lo scheletro ripropone in modo fedele la struttura neuronale, questa misura è caratterizzata da un alto grado di accuratezza.

  Data la struttura caratteristica del neurone, un altro parametro discriminante la morfologia è il numero di ramificazioni. Questo può essere calcolato sottraendo uno al numero di estremità neuritiche individuate nell'oggetto, partendo dal presupposto che i neuriti non abbiano contatti tra loro. Ma questa ipotesi non è verificata nelle colture di neuroni, in cui i neuriti sono collegati a formare falsi punti di biforcazione e spesso nascondendo le estremità dei dendriti. Per ovviare a ciò, si è utilizzato un algoritmo, implementato da **Cormen at al.** (2001), che stima il numero di ramificazioni partendo da considerazioni geometriche.

  Per il calcolo dei neuriti primari, si esegue una conta dei punti in cui dal soma cellulare partono le ramificazioni. Ciò è eseguito tecnicamente contando il numero di punti in cui lo scheletro è in contatto con il *cell body ROI*. Queste radici sono stimate scorrendo i pixel dello scheletro fino a che non è individuato un punto in comune con il *cell body ROI*. Da tale conta sono esclusi i neuriti la cui lunghezza è inferiore a una soglia empiricamente determinata.

  Infine, è possibile calcolare il *territory* di un neurone, ovvero l'area del più piccolo poligono convesso che contiene l'intero scheletro del neurone. Il poligono è ottenuto utilizzando una versione modificata di un algoritmo ideato da **Graham** (1972), mentre l'area è calcolata utilizzando gli strumenti già implementati in ImageJ.

- Dominant Neurite: contiene un algoritmo che permette di isolare il neurite dominante, ovvero il neurite primario di lunghezza massima.

Questo tipo di operazione è effettuata a mano: l'utente, infatti, seleziona con un click del mouse i punti in cui il neurite dominante è collegato alla struttura principale. Di solito, il neurite dominante è semplice da individuare, ma comunque il tool permette di selezionare più candidati se necessario.

NeuronMetrics è un tool user-friendly, gratuito e che offre ai neuroscienziati un tool per analisi morfometriche semiautomatizzate. Permette di fare *processing* e analisi di immagini in tempi relativamente brevi, e con un buon grado di accuratezza. I risultati ottenuti con NeuronMetrics sono comparabili a quelli ottenuti con altri tipi di software.

1.3.5 Sholl Analysis [12]

Sholl Analysis è un tool di Image J che automaticamente opera l'analisi di Sholl su un neurone.

L'algoritmo lavora su immagini ad 8 bit a livelli di grigio che soddisfano due caratteristiche fondamentali:

- Tutti i pixel che rappresentano la struttura neuronale siano allo stesso livello di grigio
- Tutti i pixel che non rappresentano la struttura neuronale siano ad un livello di grigio differente rispetto a quelli del punto precedente

Il neurone, quindi, deve presentarsi come un oggetto di un unico colore, diverso da quello del background. Per alcuni tipi di immagine, basta un'operazione di sogliatura, implementata in ImageJ; per immagini particolarmente rumorose, invece, potrebbe essere necessario il riconoscimento neuronale, ad esempio con NeuronJ.

Prima di usare il plug-in, è di fondamentale importanza indicare il rapporto di scala, quindi conoscere le dimensioni in μm del pixel dell'immagine da elaborare: infatti, l'algoritmo usa questa informazione per calcolare e riportare in uscita i risultati dell'analisi. Nello specifico, l'informazione di scala è il fattore di conversione tra il pixel e le unità di lunghezza, e viene settata dal menù di ImageJ (Analyze → Set Scale).

Se il rapporto di scala dell'immagine non è noto, l'analisi può essere comunque condotta usando il pixel come unità di lunghezza, ma l'utilità del risultato è limitata, perché non si possono confrontare i risultati ottenuti da immagini diverse. La scala, anche se settata, è ignorata per immagini con pixel di dimensioni anisotropiche.

L'algoritmo interno è basato sui principi dell'analisi di Sholl manuale: crea cerchi concentrici intorno al centro dell'albero neuronale (dendritico o assonale) e conta quante volte l'albero interseca la circonferenza di questi cerchi.

L'analisi di sviluppa in 3 step:

- L'utente seleziona con il "Point Selection Tool" il centro dell'oggetto da analizzare. In alternativa, usando lo "Straight Line Selection", può selezionare una linea che collega il centro al punto più distale del neurone: in questo modo, automaticamente si ottengono le coordinate del centro e il raggio del cerchio più grande.
- Avviato il plug-in, il programma mostra in una *dialog box* (Fig. 7) il valore in μm di alcuni parametri in ingresso:

1. Raggio minore: raggio del cerchio più piccolo. E' il primo cerchio sul quale si conta il numero di intersezioni con l'albero dendritico.
2. Raggio maggiore: raggio del cerchio più grande.
3. Distanza intercerchio: distanza tra i cerchi concentrici. Se l'utente setta il parametro a 0, si effettua una misura continua delle intersezioni.
4. Span del raggio: intervallo intorno al valore del raggio r per cui è effettuata la misura continua delle intersezioni. Queste informazioni, non richieste nell'analisi di Sholl tradizionale, in cui le intersezioni sono calcolate su un'unica circonferenza, sono combinate insieme per calcolare un unico valore del numero di intersezioni per quel determinato r. Se settato a 0, si effettua solo l'analisi di Sholl tradizionale, e verrà quindi effettuata un'unica misura del numero di intersezioni per quel raggio.
5. Tipo di span: attraverso un menu a tendina, l'utente può selezionare la funzione (media o mediana) con cui vuole combinare il numero di

intersezioni all'interno dell'intervallo selezionato nel punto precedente, in modo da ottenere un unico valore.

6. Fit curve and computer descriptors: se selezionato, al termine delle operazioni i dati sono fittati, altrimenti si ottengono le misure "grezze". Se selezionato lo "Show_Parameters", si ottengono nel dettaglio i risultati del *fitting*, e una stima della bontà dello stesso.
7. Create Intersection Mask: se selezionato dell'utente, il plug-in automaticamente crea e visualizza un'immagine a 32 bit raffigurante il neurone analizzato, colorato tenendo conto del profilo di Sholl calcolato.

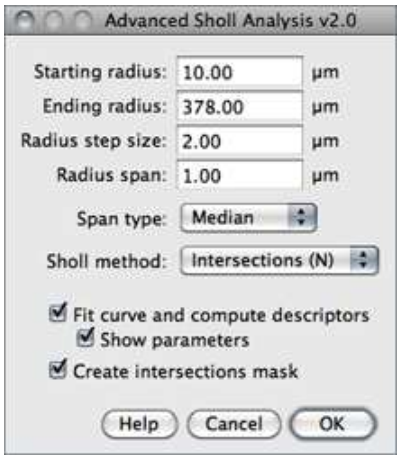


Fig. 7 Dialog-box per l'inserimento di alcuni dati in ingresso

Sono disponibili quattro diverse modalità di presentazione dei risultati:

- Intersezioni: si ottiene un grafico con il numero di intersezioni in funzione della distanza dal centro. I dati sono fittati con una polinomiale di quinto ordine (del tipo $N=a+br+cr^2+dr^3+er^4+fr^5$) . Sono automaticamente calcolati raggio per il quale ho il massimo numero di intersezioni, massimo numero di intersezioni e il rapporto tra il valor medio delle intersezioni e l'intera area occupata dall'albero dendritico.
- Intersezioni/Area: si ottiene un grafico in cui si riporta, in funzione del raggio r, il rapporto tra il numero di intersezioni e l'area del cerchio di raggio r stesso. I dati sono fittati con una funzione del tipo $N/S=ar^b$.

- SemiLog: si ottiene un grafico con il logaritmo del numero di intersezioni in funzione della distanza dal centro. Per l'interpolazione è utilizzata la regressione lineare ($log(N/S)=a+kr$), ed è calcolato automaticamente il coefficiente di Sholl come pendenza della retta ottenuta.
- LogLog: Si ottiene un grafico con il logaritmo del numero di intersezioni in funzione del logaritmo della distanza dal centro. Per fittare i dati è utilizzata una funzione esponenziale con offset ($log(N/S)=ae^{-b*log(r)}+c$).

con N numero di intersezioni, S area del cerchio e r distanza dal centro del soma.

1.3.6 FracLac [13]

FracLac è un software per analisi delle immagini sviluppato come plug-in per ImageJ. A differenza di altri plug-in è anche disponibile come applicazione Java stand-alone. E' particolarmente adatto per descrivere la geometria di strutture biologiche ramificate rappresentate in immagini binarie. FracLac, infatti, riconosce solo oggetti descritti da pixel neri su sfondo bianco. Se l'immagine disponibile non è binaria, può essere resa tale elaborandola con algoritmi già presenti in ImageJ.

Dopo aver avviato FracLac su ImageJ, appare un pannello (Fig. 8), attraverso il quale l'utente può decidere il tipo di scansione da effettuare (bottoni viola), settando alcuni parametri fondamentali per l'analisi, eseguire la scansione stessa (con i bottoni azzurri l'utente decide se eseguirla sull'intera immagine o su una parte di essa) e infine salvare i risultati.

FracLac esegue quattro diversi tipi di analisi:

- Dimensione frattale: viene calcolato il parametro D_B con la tecnica del *box-counting*. In sintesi, si disegnano automaticamente sull'immagine da analizzare dei quadrati di lato l crescente, a formare una griglia sull'immagine stessa. Per ogni l si conta il numero di maglie occupate dall'oggetto. Si ottiene un grafico del logaritmo del

numero di maglie occupate in funzione del logaritmo di l. La pendenza di tale curva è D_B, indice della frattalità dell'oggetto.

- Lacunarità: il plug-in ritorna in modo automatico l'indice di lacunarità Λ, definita come la variazione della densità dei pixel a differenti l;
- Multi-frattalità: è possibile reperire, in biologia, strutture in cui non c'è un'unica regola di *scaling*, bensì più regole. In altri termini, la struttura risultante è ottenuta da un insieme di monofrattali "sovrapposti". Questi non possono essere analizzati con uno scanning tradizionale: FracLac implementa *ad hoc* lo scanning multi-frattale;
- Forma e dimensioni dei pattern dell'oggetto frattale: oltre agli indici di frattalità e lacunarità, FracLac permette di ottenere misure sulla forma degli oggetti, come anche sulla densità dei pixel in un'immagine. Sono parametri, quali ad esempio area e perimetro del poligono che contiene l'oggetto, calcolati solo se espressamente richiesti dall'utente;

Per ognuna delle analisi elencate, l'utente deve settare, prima di avviare l'operazione, alcuni parametri, come ad esempio il valore minimo e massimo di l: ciò è possibile grazie a delle *dialog-box*.

E' possibile scansionare l'immagine in quattro modi diversi:

- *regular box-counting*, descritto precedentemente a proposito del calcolo della dimensione frattale;
- *subareas scan*, che permette di operare lo scanning solo su una determinata parte dell'immagine, quindi di individuare la frattalità locale;
- *sliding box lacunarity*, misura dell'eterogeneità in un'immagine digitale. Posizionato sull'immagine un quadrato di lato l noto, si contano il numero di pixel all'interno di esso. Questa grandezza è calcolata spostando il dapprima orizzontalmente di un certo x, e poi verticalmente di un certo y. Ciò è ripetuto per valori crescenti del lato l del quadrato. Il risultato differisce dal tradizionale metodo box-counting, in quanto è possibile una sovrapposizione delle maglie costituenti la griglia;

- *multifractal scan*: nel caso di strutture multi-frattali, FracLac genera una distribuzione di massa (ossia il numero di pixel per box) per un'immagine. Da questa, è possibile calcolare la dimensione generalizzata D_Q, indice della variazione di massa con la dimensione dei box ε in un'immagine. FracLac infine mostra il grafico di D_Q in funzione di Q (Q è un esponente usato nel calcolo degli spettri multi frattali, i cui valori minimo e massimo sono indicati dall'utente).

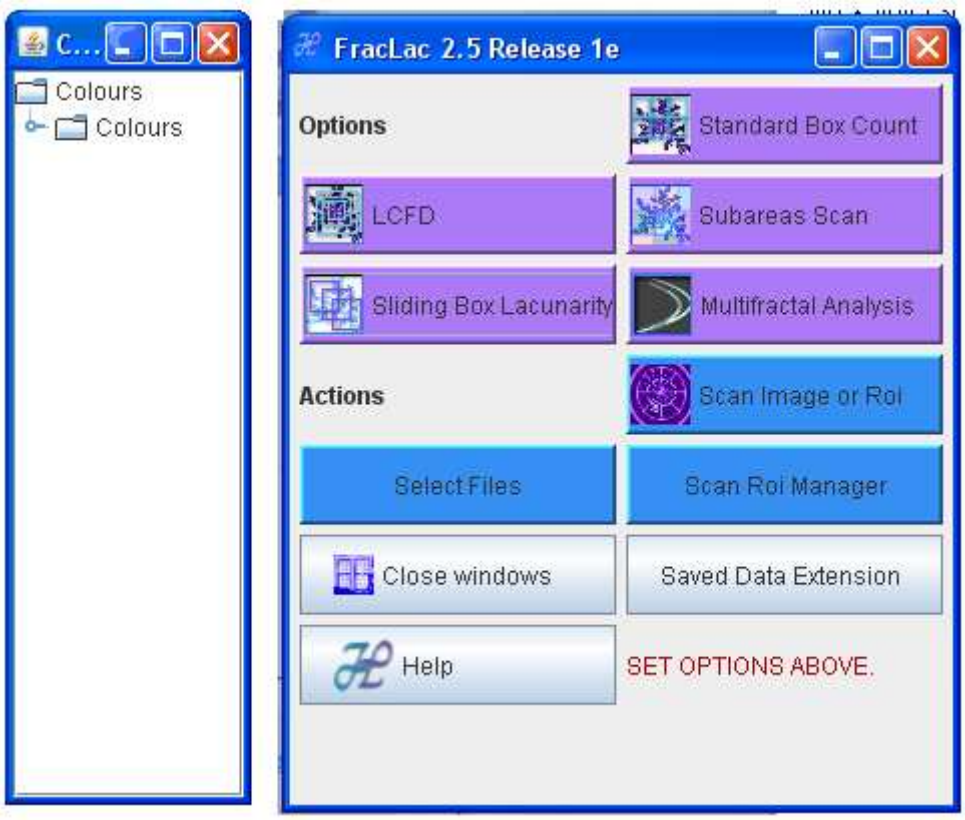


Fig. 8 Interfaccia grafica di FracLac

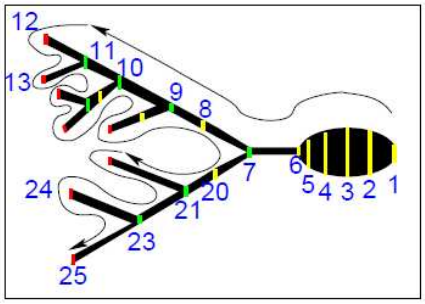
1.3.7 Neuron_Morpho [2]

Neuron_Morpho è un plug-in di Image J che, dato uno stack di immagini ottenute tramite microscopia confocale, permette all'utente di conoscere le coordinate x, y e z di un punto selezionato dall'utente, oltre ad altre informazioni utili a ricostruire la morfologia neuronale. Alla fine dell'elaborazione, il programma ritorna una serie di misure salvate in formato swc. Prima di utilizzare il plug-in, è necessario eseguire il *load* dello stack di immagini. Nel caso in cui le immagini contengano pochi dettagli, è possibile creare uno stack selezionando

regioni più piccole, o limitando il numero di *slice* sovrapposte. A questo punto, premendo "Start" sulla *dialog-box* di Neuron_Morpho, con lo strumento "*line*" di ImageJ l'utente traccia una linea attraverso la sezione del neurone, come in Fig. 9. L'algoritmo calcola la lunghezza della linea tracciata, e ne calcola il punto medio, infine organizza le informazioni che estrae da tale punto in forma matriciale:

- Number: numero intero via via crescente che identifica il punto;
- Type: numero intero compreso tra 0 e 7. Ad ogni numero corrisponde un certo tipo di segmento neuronale, ad esempio, sinapsi, soma, assone, dendrite o punto di biforcazione. La cifra 0 indica che l'algoritmo non è riuscito nell'identificazione;
- x, y, z: rappresentanti le tre coordinate del punto del neurone selezionato;
- Radius: raggio della sezione, dividendo per due la linea che l'utente traccia sull'immagine;
- Previous: numero intero crescente che identifica il punto precedente a quello che si sta considerando, muovendosi dal soma all'estremità del neurone;

  Questo procedimento si itera per ogni punto che l'utente vorrà considerare: la tabella quindi si compila automaticamente riga per riga. Ad analisi ultimata, l'utente può salvare la matrice cliccando sul tasto "*Save*". Inoltre, il programma permette di salvare ulteriori righe di informazioni su una matrice pre-esistente grazie al tasto "*Open*".



Number	Type	x	y	z	Radius	Previous
1	1	508.50	199.00	2.00	15.00	-1
2	1	489.00	199.50	2.00	25.50	1
3	1	468.00	200.00	2.00	31.00	2
4	1	439.00	199.50	2.00	31.50	3
5	1	412.00	200.50	2.00	25.50	4
6	3	391.00	198.50	2.00	5.50	5
7	5	328.00	199.00	2.00	6.00	6
8	3	265.50	163.00	1.00	8.01	7
9	5	218.00	137.00	1.00	7.00	8
10	5	157.50	106.00	1.00	7.02	9
11	5	113.00	80.50	1.00	6.50	10
12	6	65.00	55.50	1.00	6.50	11
13	6	63.00	105.00	1.00	5.00	11
14	3	132.50	126.50	1.00	8.01	10
15	5	114.00	138.50	1.00	8.50	14
16	6	81.50	123.50	1.00	7.51	15
17	6	89.50	169.50	1.00	6.52	15
18	3	182.00	156.50	1.00	6.57	9
19	6	149.50	169.00	1.00	6.02	18
20	3	281.50	227.50	3.00	6.52	7
21	5	241.00	249.50	3.00	7.50	20
22	6	147.00	211.00	3.00	4.00	21
23	5	178.50	287.00	3.00	6.02	21
24	6	96.00	254.50	3.00	7.50	23
25	6	99.50	336.50	3.00	5.52	23

Fig. 9 Finestra del plugin Neuron_Morpho. Applicazione a un neurone stilizzato.

Lo svantaggio maggiore nell'utilizzo del plug-in è l'intervento continuo dell'utente: infatti, si ottengono informazioni solo delle zone dell'oggetto selezionate manualmente. Di contro, è un plug-in gratuitamente distribuito, che può essere utilizzato su vari tipi di sistemi operativi, e, come dimostrato da **K. M. Brown et al.** (2005) [2], i risultati morfologici ottenuti sono comparabili con quelli estraibili con Neurolucida.

1.3.8 Cell counter

Per quanto riguarda la conta cellulare effettuata con Image J, ci sono due modi per ottenerla: uno manuale ed uno automatico.

Il metodo manuale, che richiede l'istallazione di un plug-in per essere utilizzato, permette all'utente di contare le cellule in una determinata immagine. Il conteggio è effettuato al click del mouse: infatti, la cellula è marcata con un quadratino colorato ad ogni click, e contemporaneamente si incrementa la variabile "conta" in maniera automatica. Con questo plug-in è possibile elaborare immagini sia a colori che a livelli di grigio, anche se per quest'ultime i marker non

sono colorati ma neri, quindi è più difficile distinguerli. Il plug-in dà la possibilità, nel caso in cui siano presenti più tipi cellulari all'interno della stessa fotografia, di avere più marker di colore diverso. Inoltre, nel caso di errore, è possibile rimediare cliccando sul tasto "Undo", che elimina l'ultima selezione. Quando l'utente è sicuro di aver selezionato tutte le cellule presenti nell'immagine, selezionerà il tasto "Results" e otterrà, quindi, il numero delle cellule stesse. Il risultato può essere salvato in un file .xls per successive elaborazioni su foglio elettronico.

A volte, però, gli oggetti da contare all'interno di un'immagine sono tanti, e il metodo manuale non è quindi molto efficiente: per questo è disponibile anche un metodo che effettua la conta in maniera automatica.

Il metodo automatico non richiede l'uso di particolari plug-in, ma si avvale degli algoritmi per l'*image processing* già implementati all'interno di ImageJ. Il metodo è pensato per le immagini a livello di grigio: se si dispone, quindi, di un'immagine a colori, bisogna convertirla in scala di grigi. A questo punto, si opera un'operazione di sogliatura, il cui risultato è un'immagine binaria, che permette di evidenziare in maniera netta gli oggetti da contare rispetto al background. Nel caso ce ne sia bisogno, è possibile eliminare rumore di background oppure separare due cellule che nell'immagine risultano unite con opportuni tool presenti in ImageJ. Quando l'utente è soddisfatto dell'immagine binaria ottenuta, può cliccare sulla barra dei menù di Image J su Analyze → Analyze Particles. Si aprirà una finestra, in cui l'utente indicherà le dimensioni minima e massima delle cellule da contare e la forma delle stesse (Fig. 10). L'algoritmo automaticamente crea una copia dell'immagine originale, i cui oggetti sono labelizzati da numeri via via crescenti, e visualizza una finestra che dà informazioni non solo sul numero di oggetti, e quindi di cellule, ma anche sull'area totale occupata dagli stessi e sull'area media di ogni particella.

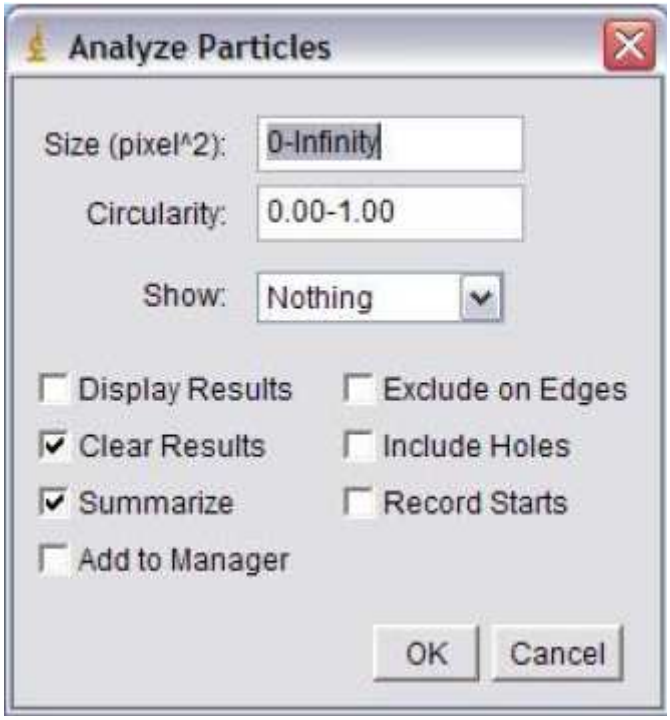


Fig. 10 Conteggio automatico con ImageJ

1.4 NeuronStudio [14][15][16]

NeuronStudio (Fig. 11) è un software gratuito pensato per ricostruire le strutture neuronali partendo da immagini al confocale. Il programma è stato implementato dal Computational Neurobiology and Image Center (CNIC) di New York per ottenere in modo automatico il *tracing* della struttura neuronale da *stack* di immagini. Infatti, NeuronStudio permette, una volta caricata un'immagine, di:

- Tracciare automaticamente tutti i dendriti connessi al neurone che si vuole studiare. Questa operazione è implementata a partire da un pixel, detto *seed*, indicato dall'utente e appartenente al neurone. I segmenti dendritici disconnessi dal corpo neurale a causa di errori nell'acquisizione dell'immagine o nel *processing* possono essere tracciati uno alla volta inserendo in ogni ramo neuronale un seed;
- Eseguire il processing delle immagini: sono implementati, infatti, i filtri più comuni per migliorare la qualità delle immagini, aumentandone il contrasto e la nitidezza dei contorni;
- Eseguire misure di lunghezza su una certa struttura, che l'utente ottiene cliccando con il mouse un'estremità dell'oggetto da

misurare e rilasciandolo sull'estremità opposta. Il risultato dell'operazione comparirà automaticamente a display;

- Analizzare le immagini. Infatti, NeuronStudio permette di effettuare l'analisi di Sholl in maniera automatica. L'utente inserisce la distanza inter-cerchio, mentre il minimo e il massimo raggio, se non indicati dall'utente, sono calcolati direttamente dal software. Al termine dell'analisi, è visualizzata una tabella con i valori morfologici ottenuti, in particolare lunghezza totale e volume totale dei dendriti e numero di biforcazioni all'interno dell'albero dendritico;
- Salvare l'immagine dopo il *processing* o il *tracing*, in vari formati tra i più comuni, o i dati, organizzati sotto forma matriciale, in cui sono stati salvati i parametri misurati dall'utente.

La grande novità di questo software, rispetto ai precedenti, è che permette la ricostruzione di neuroni in tre dimensioni. La quantificazione precisa delle morfologia in 3D è essenziale per conoscere le funzioni dei neuroni, il loro sviluppo la loro plasticità e le loro disfunzioni in caso di patologie neurodegenerative. NeuronStudio implementa un nuovo approccio per il riconoscimento e all'analisi della struttura neuronale, partendo da immagini al confocale, più veloce ed accurato delle tecnologie semiautomatiche esistenti in rete. La rappresentazione digitale della morfologia neuronale da immagini è infatti in genere effettuata tramite *tracing* manuale, è quindi soggetta a molti errori ed è poco riproducibile. I problemi dei software automatici, come anche AutoNeuron, discusso precedentemente, consistono nella difficoltà nella caratterizzazione delle spine dendritiche in modo automatico, ma anche nella scarsa precisione, dovuta agli errori di quantizzazione e ai metodi di scheletrizzazione.

Inoltre, sofisticate tecniche di visualizzazione 2D e 3D facilitano la verifica della corretta ricostruzione del neurone. Il risultato dell'elaborazione ottenuta con NeuronStudio è compatibile con gli standard, sia dei software per la modellizzazione neuronale che per l'analisi morfologica.

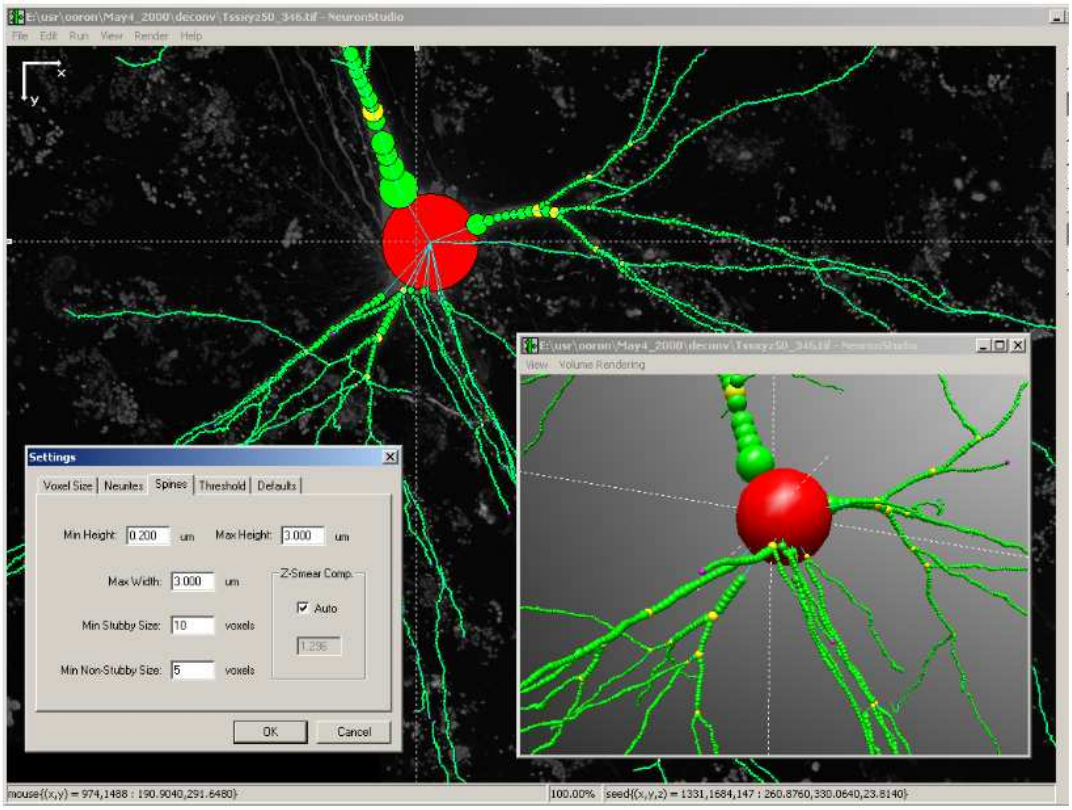


Fig. 11 Interfaccia grafica NeuronStudio

1. 5 MetaMorph [17][18]

Il sistema di imaging MetaMorph, sviluppato dalla *Molecular Devices Corporation* è considerato uno standard a livello industriale per l'acquisizione automatizzata di immagini tramite microscopio, per il controllo del dispositivo e l'analisi delle immagini stesse. Infatti, negli ultimi 25 anni, è stato molto utilizzato nei laboratori di tutto il mondo, per le sue capacità di acquisizione, analisi e processing, contribuendo ad importanti scoperte sulla morfologia e la funzione cellulare. E' caratterizzato dall'essere un software molto flessibile e versatile, che esegue molteplici operazioni, dall'acquisizione multidimensionale alla ricostruzione 3D all'ottenimento di misure morfologiche. MetaMorph è stato utilizzato in molti studi, reperibili in letteratura, per la sua velocità e flessibilità.

E' una piattaforma modulare, i cui moduli permettono:

- Il controllo e l'acquisizione immagini tramite videocamera;
- L'automazione e il controllo del microscopio;

- Il posizionamento e l'illuminazione del campione;
- La semplificazione di task ripetitivi e complessi.

Il sistema di imaging MetaMorph è una potente ed efficace integrazione tra software e hardware, che permette ai ricercatori di acquisire e analizzare immagini al microscopio. Per quanto riguarda il controllo del dispositivo, MetaMorph integra in un'unica workstation l'hardware per la microscopia e le varie periferiche. Il controllo è eseguito sulle camere a tecnologia CCD ad alta risoluzione, sul microscopio, sui filtri, sulla parte robotica del dispositivo, e sul piezoelettrico per la messa a fuoco. Il software permette di eseguire delle analisi sulle immagini acquisite. Infatti, dopo l'acquisizione, i ricercatori possono utilizzare MetaMorph per misure e analisi. L'utente definisce le regioni da analizzare e il software calcola angoli, distanze e aree. E' possibile effettuare anche misure sensitometriche e di intensità. Oltre alle operazioni comuni anche ad altri tool, come la sogliatura, il miglioramento del contrasto e il filtraggio, sono disponibili algoritmi più sofisticati, come il modulo di analisi morfometrica IMA, acronimo di *Integrated Morphometry Analysis*. IMA usa un'interfaccia grafica molto semplice da usare (Fig. 11): permette, infatti, di visualizzare le varie misure effettuabili su una determinata immagine (angoli, distanze, aree etc.), di "etichettare" le varie regioni o oggetti presenti nell'immagine stessa attraverso numeri che la identifichino e, nel caso di più immagini in stack, di compiere più misurazioni contemporaneamente.

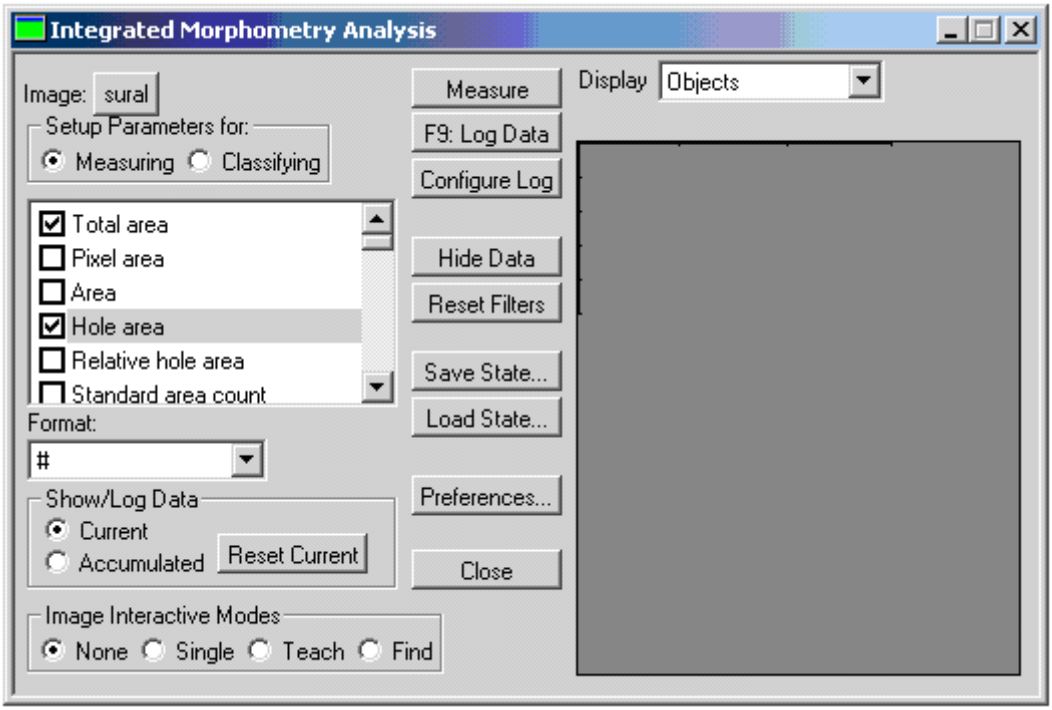


Fig. 11 Interfaccia grafica IMA

Attraverso operazioni quali sogliatura, o operazioni morfologiche l'utente rende l'immagine idonea alla successiva analisi morfologica. Tali operazioni prevedono un preliminare miglioramento del contrasto dell'immagine, e poi varie operazioni di *processing*, quali deconvoluzione 2D, ricostruzione dell'immagine in 3D, filtri che rendano più nitidi i contorni degli oggetti, campitura dello sfondo, rilevamento dei contorni, convoluzione, segmentazione, dilatazione, erosione, scheletrizzazione, ma anche FFT o rimozione manuale di singoli pixel indesiderati. Per quanto riguarda l'analisi delle immagini così elaborate, oltre a distanze, calcolo delle coordinate in 3D e angoli, il sistema classifica gli oggetti in base alla loro forma, effettua misure di intensità (fondamentali per lo studio della motilità proteica) e densitometriche, permette di visualizzare i risultati sotto forma di istogrammi, o di salvare le misurazioni all'interno di una matrice di dati, elaborabili con Excel. Più specificatamente, permette, nel caso in cui si abbiano *stack* di immagini, di allineare lo *stack* stesso, di ricostruire la struttura in 3D, fare misure in contemporanea su tutti i piani, di evidenziare particolari strutture, di creare mappe topologiche di superficie, di selezionare uno o più piani su cui effettuare l'analisi, e di visualizzarli. Inoltre, il software permette di seguire il movimento

di una determinata sostanza (molti studi utilizzano proteine fluorescenti che si legano a molecole di superficie delle cellula, o ai microtubuli, acidi nucleici, o lipidi): i ricercatori possono così misurare le coordinate X e Y di tali particelle, la loro velocità e i loro spostamenti, visualizzando i risultati tramite grafici.

Usando questo software, è possibile sviluppare vari protocolli di analisi, utilizzabili in diverse aree di ricerca: uno tra tutte, interessante dal punto di vista neuro morfologico, è lo studio di **Gensel** (2010), che utilizza MetaMorph per eseguire l'analisi di Sholl e quantificare la crescita neuronale utilizzando diversi fattori di crescita. E' un metodo valido poiché è caratterizzato da assenza di *bias*, da un buon livello di accuratezza e tempi ridotti di esecuzione rispetto all'analisi di Sholl effettuata manualmente.

1.6 TREES toolbox [19][20][21]

Implementato in ambiente Matlab, TREES toolbox permette di ricostruire automaticamente l'arborizzazione dendritica da immagini ottenute tramite microscopia, di visualizzare e analizzare l'albero dendritico, di comparare la struttura di neuroni diversi e di comprendere in che modo la struttura dell'albero dendritico sia legata all'ottimizzazione della conduzione dei segnali nervosi. Infatti, è un tool che oltre alla ricostruzione dell'albero neuronale, alla visualizzazione e all'editing di neuroni virtuali, effettua anche un'analisi elettrotonica, cioè calcola parametri relativi alle proprietà dei neuroni di modificarsi dal punto di vista dell'eccitabilità e della conduzione a causa del passaggio stesso di corrente. E' un'applicazione gratuita, di cui è scaricabile anche il codice sorgente: in questo modo, gli utenti possono contribuire alla stesura del tool, eliminando bug o aggiungendo algoritmi.

Più di un secolo fa **Ramon y Cajal** descrisse l'arborizzazione dendritica in tutte le sue forme e varianti, e spiegò come parametri fisici, ad esempio spazio, volume citoplasmatico e tempi di conduzione, condizionino la morfologia cellulare. Queste informazioni possono essere implementate in un software che genera

neuroni virtuali o intere reti neurali sintetiche, ma possono essere utilizzate anche per la ricostruzione *model-based* di neuroni, a partire da immagini ottenute al microscopio confocale.

Il principio alla base degli algoritmi implementati in questo tool è che la struttura della cellula nervosa è approssimabile a un grafo che connette un set di nodi, rappresentanti i punti di biforcazione all'interno dell'albero neuronale o, più in generale, punti particolarmente importanti per descrivere la geometria del neurone stesso. Ai nodi, connessi tra loro da linee detti "archi", sono assegnati dei numeri crescenti. E' importante che la regola di etichettatura dei nodi sia univoca, soprattutto per poter operare un confronto tra grafi, sia topologicamente che elettrotonicamente. Il collegamento tra nodi è codificato, anch'esso in modo univoco. Alcune proprietà, come la distanza dei rami dalla radice o la lunghezza dei vari segmenti che compongono le ramificazioni, possono essere dedotti direttamente dalla rappresentazione a grafo. Per collezionare tutte le informazioni relative all'albero dendritico, gli algoritmi lavorano in modo ricorsivo.

TREE-toolbox permette:

- Ricostruzione manuale: è presente un'interfaccia grafica integralmente dedicata alla ricostruzione manuale dell'albero neuronale, per ogni immagine che costituisce lo stack;
- Tree editing: strumento che permette di ricreare l'arborizzazione dendritica di un singolo neurone, ma anche la connettività di più neuroni, a costituire una rete neurale più o meno complessa;
- Analisi elettrotonica: calcolate allo stato stazionario, le proprietà elettrotoniche, cioè riguardanti le modificazioni dell'eccitabilità e della conduzione del nervo determinate dal passaggio di corrente elettrica nello stesso sono, ad esempio, resistenza di ingresso locale.;
- Branching statistics: permette di calcolare i parametri morfologici più tradizionali, come il centro di massa, o la lunghezza dei path.

Si riassumono le caratteristiche dei software elencati nella tabella di Fig. 12.

Software		Sholl Analysis	Fractal Dimension	Soma Area	Neurite length	Neuron count	3-way PCA	Type of cells	Principal Advantages	Principal defects
Neurolucida		Yes	No	No	Yes	Yes	No	Not only neurons; Fluorescent or not.	Good resolution;	Not free; Only Windows.
Syn-D		Yes	No	Yes	Yes	Yes	No	Fluorescent	Free; Automated; Time-efficient; Precise.	Only Windows and MacOS.
ImageJ	**NeuronJ**	No	No	No	Yes	No	No	All neurons.	Free	Only tracing, no analysis.
	NeuronTracer	No	No	No	Yes	No	No	Specific marker; Confocal images.	Free; Totally automated.	Only tracing, no analysis.
	Neurphology	No	No	Yes	Yes	No	No	All neurons.	Free; Automated; Time-efficient.	Incompleted analysis
	Neuron Metrics	No	No	Yes	Yes	No	No	All neurons.	Free; SemiAutomated.	Incompleted analysis
	Sholl Analysis	Yes	No	No	Yes	No	No	All neurons.	Free; Automated.	Only Sholl Analysis.
	FracLac	No	Yes	No	No	No	No	All neurons.	Free; Automated.	Only Fractal Dimension.
	Neuron_Morpho	No	No	No	Yes	No	No	Confocal images	Free;	Not Automated
	Cell counter	No	No	No	No	Yes	No	All images	Free;	Only count;
Neuron Studio		Yes	No	No	Yes	No	No	Confocal images	Free; 3D reconstruction.	Only confocal Images.
MetaMorph		Yes	No	Yes	Yes	No	No	All neurons	Accurated.	Not free;
TREE toolbox		No	No	No	Yes	No	No	All neurons.	Good neurite reconstruction.	Need Matlab; Only tracing.

Capitolo 2

Approccio neuroscientifico allo studio delle alterazioni della morfologia neuronale associate a diversi tipi di patologie

L'implementazione di software, come Ne.Mo., per la caratterizzazione morfologica e la conta di neuroni *in vitro* durante le prima fasi di sviluppo del cervello è di fondamentale importanza per ottenere misure che siano ripetibili e accurate in tempi ragionevolmente brevi. La procedura di analisi sviluppata in Ne.Mo. è stata pensata per lo studio della patogenesi dell'autismo, ma può essere applicata a qualsiasi patologia correlata a cambiamenti nella morfologia e nel numero di neuroni.

In questo capitolo verrà presentato un quadro generale sulle patologie e, più in generale, sulle alterazioni dello stato fisiologico che portano a modificazioni della struttura neuronale. Gli studi, condotti su tessuti *post-mortem* o *in vivo* tramite le nuove tecniche di *imaging* non invasivo, hanno dimostrato alterazioni sia macroscopiche, come alterazioni del volume cerebrale, che microscopiche, riguardanti, quindi, la struttura della singola cellula, dall'area del soma alla lunghezza dei neuriti.

2.1 Disordini dello spettro autistico [22] [23][24][25]

L'autismo è un disordine che interessa le funzioni cognitive, linguistiche e comportamentali di un individuo, ed è il risultato di influenze ambientali e genetiche. Il primo a descriverne le caratteristiche fu **Kanner** (1943), che lo definì come una sindrome che si manifesta entro i 36 mesi di vita, caratterizzata da problemi cognitivi, linguistici e nelle interazioni sociali, risposta anormale agli stimoli sensoriali, contatti visivi quasi del tutto assenti con l'interlocutore, singolare capacità di memorizzazione, comportamenti ripetitivi e stereotipati; la presenza fisica non è compromessa, a differenza di altre sindromi, come quella di Down. Colpisce circa un bambino su 150, in prevalenza di sesso maschile. E' comprovato che l'autismo sia caratterizzato da un ampio spettro di manifestazioni cliniche, che lo rendono assai eterogeneo. Ciò lo rende difficile da concettualizzare come un meccanismo neurologico univoco che possa racchiudere tutte le caratteristiche del disordine stesso. Se si considera questa eterogeneità, si comprende il perché si parla di "autismi", o più tecnicamente di "disordini dello spettro autistico" (Fig. 1)

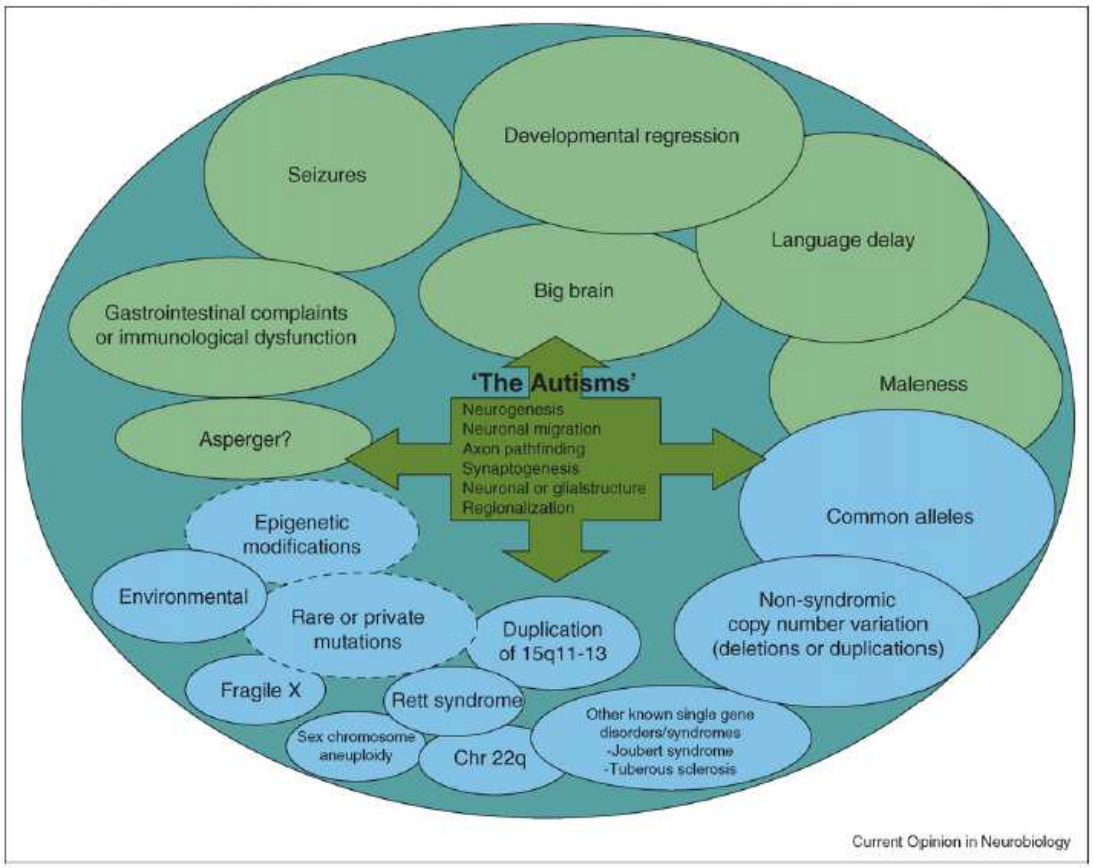


Fig. 1 Schema dei disordini dello spettro autistico [26].

Basandosi sulle osservazioni cliniche di individui colpiti dal disordine dello spettro autistico, si sono identificate delle aree che presentano delle alterazioni se confrontate con soggetti di controllo: sistema limbico, lobi mediali temporali, talamo, gangli della base, sistema vestibolare e verme cerebellare. Tuttavia, l'identificazione nel dettaglio del substrato neuro-anatomico in un cervello autistico è ancora incompleta, a causa della scarsità delle risorse patologiche umane disponibili, e la scarsità di modelli animali.

Per quanto riguarda le caratteristiche macroscopiche della patologia, la maggior parte degli studi anatomici sul disturbo autistico si concentrano su bambini, adolescenti o adulti, pochi su neonati, quindi sono poche le informazioni inerenti allo sviluppo del cervello dopo la nascita. Indicatore della grandezza cerebrale è la circonferenza della testa, semplice da misurare anche su bambini molto piccoli. Il volume cerebrale invece è calcolato usando la risonanza magnetica (MRI), o pesando il cervello *post mortem*. Alla nascita, la circonferenza media di un paziente autistico è approssimativamente normale, come si

evince dagli studi di **Courchesne e Pierce** (2005). A quattro anni, gli individui autistici presentano un cervello più grande del normale di circa il 10% se comparati agli individui di controllo (Fig. 2): è il risultato che hanno ottenuto **Courchesne et al.** nel 2001, **Sparks et al.** nel 2002 e **Redcay e Courchesne** nel 2005 basandosi su studi in vivo effettuati tramite MRI e su analisi autoptiche. Più recentemente, il team di **Harzlett**, osservando un campione relativamente numeroso (51 bambini tra i 18 e i 35 mesi), riscontrò un incremento del solo 5%. Intorno ai sei anni, le differenze nelle dimensioni cerebrali sono meno marcate, secondo gli studi di **Courchesne** (2001, 2004) [27][28].

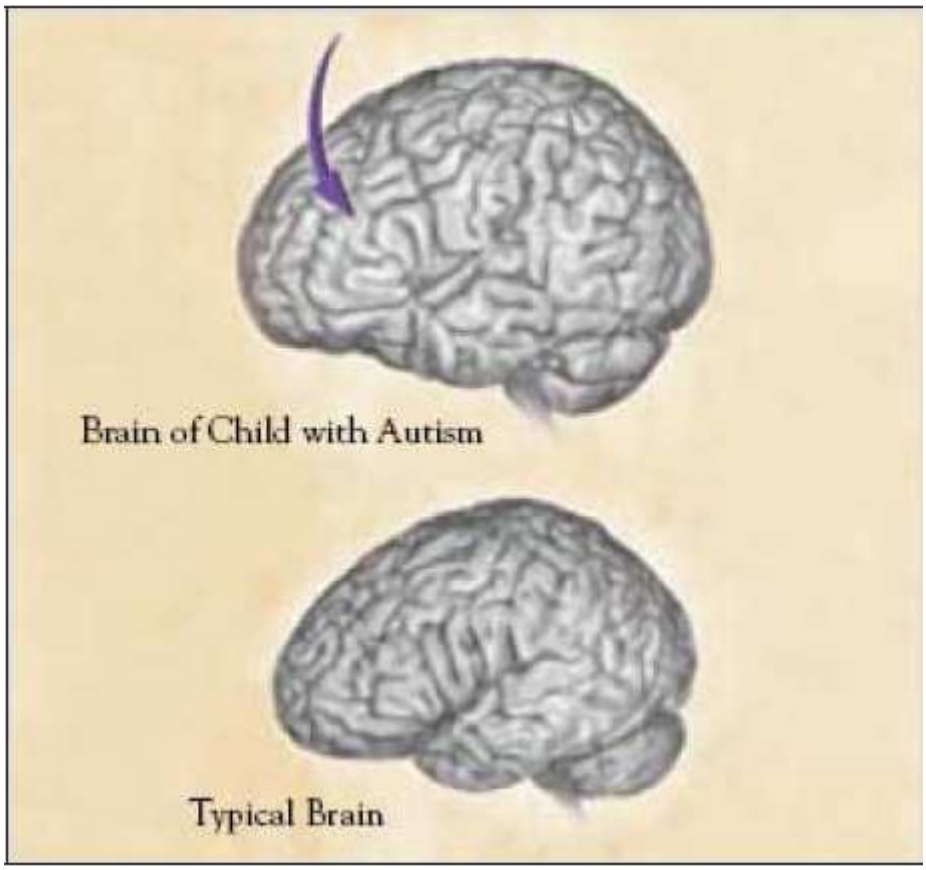


Fig. 2 Aumento del volume cerebrale in un soggetto autistico

A livello tissutale, ciò si riflette in un aumento sia della sostanza bianca che grigia cerebrale (**Carper et al.**, 2002 e **Harzlett et al**., 2005), in particolar modo di quella bianca immediatamente sottostante la corteccia cerebrale. C'è anche un aumento della sostanza bianca e grigia cerebellare, come riscontrato da **Courchesne et al.** nel 2001.

Dal punto di vista microscopico, sono reperibili vari studi in letteratura, riguardanti diverse aree cerebrali.

Rodier et al. (1996), esaminando il tronco dell'encefalo, mostrarono una marcata riduzione nel numero di neuroni.

In un recente studio autoptico, **Guerin et al.** (1996) evidenziarono un assottigliamento delle meningi e del corpo calloso.

Nel 1980, **William et al.** esaminarono materiale autoptico ottenuto da quattro individui autistici, e riscontrarono gliosi e un minor numero di cellule rispetto ad individui sani. Non si osservarono altre alterazioni gravi. Nel 1985, **Baumann e Kemper** analizzarono il cervello di un individuo autistico di 29 anni, mettendolo in relazione a un controllo dello stesso sesso e della stessa età. Le differenze più evidenti tra i due si riscontrarono nel sistema limbico e nei circuiti cerebellari. **Coleman et al.** (1985) dopo un'attenta osservazione dei circuiti neuronali e gliali nelle regioni corticali, non riscontrarono nessuna anomalia, né dal punto di vista morfologico né nella conta cellulare. Nel 1998, **Bailey et al.** notarono malformazioni a livello neocorticale in materiale autoptico. In quattro dei sei casi studiati, la corteccia cerebrale appare più sottile, ci sono aree con maggiore densità neuronale, un maggior numero di neuroni nei vari strati corticali e con orientamento alterato delle cellule piramidali. Inoltre, in uno studio più recente, **Casanova et al.** (2002) dimostrarono che nella corteccia cerebrale degli individui autistici studiati *post mortem*, nove pazienti di cui sette maschi e due femmine e undici pazienti di controllo di cui sette maschi e quattro femmine, c'è un numero maggiore di colonne, ma sono più piccole in diametro e più compatte rispetto ai soggetti normali: ciò porta a non avere differenze sostanziali nelle densità. Il risultato si è ottenuto attraverso un indice, il *gray level index,* definito come rapporto tra le aree colorate con la colorazione di Nills e le aree non colorate [29]. Allo stato attuale, non è ancora chiaro se le anormalità corticali sono da annoverare come significative nel disordine autistico. Importante sfida della ricerca futura sarà proprio capire come queste differenze osservate siano correlate all'eterogeneità clinica dei soggetti studiati. Dal 1985 ad oggi, si sono documentati altri casi, studiati con le medesime tecniche descritte precedentemente. In relazione ai soggetti di controllo, non si sono

riscontrate grandi differenze, ad eccezione della dimensione cellulare minore e dell'aumento della densità neuronale nel giro cingolato anteriore. Infatti, aree del proencefalo che risultano alterate includono l'ippocampo, il subiculum (parte inferiore della formazione dell'ippocampo), la corteccia entorinale, l'amigdala, il corpo mammillare, il giro cingolato anteriore e il setto, tutte strutture appartenenti al sistema limbico. Queste aree, rispetto ai controlli, mostrano cellule più piccole e con minore complessità nell'arborizzazione dendritica. Nell'amigdala, l'aumento più significativo della densità cellulare è riscontrabile nella zona mediale.

Diminuzione della grandezza neuronale e, di contro, aumento della densità rispetto alla normale fisiologia cellulare sono stati riscontrati anche nell'area di Broca. **Kemper e Baumann** (1998), infatti, mostrarono che in questo nucleo i neuroni sono adeguati in numero, ma più piccoli in dimensioni nel cervello degli individui autistici con meno di tredici anni; al contrario le cellule della stessa area appaiono più piccole e ridotte in numero in tutti gli individui autistici osservati con età superiore ai ventuno anni.

Al di fuori del sistema limbico, le alterazioni più visibili si riscontrano nel cervelletto e nell'oliva inferiore. Oltre a riscontrare nel 20% dei casi macrocefalia (**Kanner**, 1943), la cui causa è ancora ignota, microscopicamente vari studi attestano una diminuzione del numero delle cellule di Purkinje. Tutti gli individui autistici analizzati da **Arin et al.** (1991), infatti, presentano una significativa diminuzione del numero di queste cellule, soprattutto nella corteccia neocerebellare postero-laterale e nell'adiacente corteccia archicerebellare. Stessi risultati sono riportati da **Ritvo et al.** (1986) e più recentemente da **Baumann e Kemper** (1996) e **Bailey et al.** (1998). Il primo team, all'interno di un progetto di ricerca riguardante le tecniche autoptiche, analizzò quattro cervelli di maschi autistici tra i dieci e i ventidue anni, mettendoli in relazione con tre cervelli di maschi di controllo tra i tre e i ventitré anni e una femmina di cinque anni. La conta cellulare mostrò una significativa diminuzione delle cellule di Purkinje nell'emisfero cerebellare [30]. Baumann e Kemper, invece, usando

tecniche stereologiche su cellule di Purkinje di quattro individui di controllo e sei autistici, mostrano riduzione del numero dei suddetti neuroni, ma non significative differenze nella densità neuronale [31]. Il team di Bailey ottiene gli stessi risultati studiando sei cervelli di soggetti autistici: oltre alla macrocefalia, riscontrata in quattro casi su sei, si è osservato un ridotto numero di cellule di Purkinje, spesso accompagnato da gliosi [32]. Di contro, **Coleman et al.** (1985) non evidenziarono differenze nel numero di cellule di Purkinje, né di cellule gliali, confrontando cervelli di autistici e controlli. Nonostante l'ipoplasia e l'iperplasia riscontrata grazie alla risonanza magnetica da **Courchersne et al.** (1994) nel verme cerebellare, non si sono riscontrate, dagli studi effettuati da **Baumann e Kemper** (1996), alterazioni nel numero o nella dimensione delle cellule di Purkinje in questa regione. Di contro, si legge nelle pubblicazioni di **Hashimoto et al.** (1995), **Courchesne et al.** (2001) e **Kaufmann et al.** (2003) che il verme cerebellare, in cui è fisiologicamente predominante la sostanza grigia, si riduce in dimensioni. Per quanto riguarda la morfologia delle cellule di Purkinje, l'analisi di **Fatemi et al.** (2002) si basa fondamentalmente su due grandezze: densità e dimensioni. Utilizzando sezioni di cervelletto di vario sesso ed età di cinque individui autistici e cinque sani è stata osservata una riduzione della dimensione media delle cellule di Purkinje maggiore del 50%. Non sono state riscontrate, invece, differenze significative per quanto riguarda la densità cellulare [33].

Alterazioni sono presenti anche dal punto di vista della migrazione neuronale: studiando, infatti, cervelletto, ippocampo e zone subcorticali e periventricolari di quattro individui autistici, **Weigel et al.** (2010) hanno osservato eterotopie, che riflettono un'anormale spostamento dei neuroni [34].

Studi più recenti individuano un'altra area probabilmente coinvolta nella sindrome autistica: il claustro. Questa è una struttura corticale che fa parte della sostanza grigia, situata nella corteccia dell'insula, in continuità anteriormente con l'amigdala, e addossata alla corteccia insulare, dalla quale è separata soltanto per interposizione della

capsula esterna; medialmente, la capsula esterna lo separa invece dal *putamen*. Il suo significato funzionale non è ancora del tutto noto, ma ha connessioni con la *neocortex* [35]. A causa delle sue connessioni biunivoche con il cervello, è implicato nella coscienza e in altre funzioni di ordine superiore, quali il comportamento e le emozioni: è per questo che è oggetto di analisi per la comprensione del suo ruolo nella neurobiologia dell'autismo. Si è osservato volumi minori del claustro negli individui autistici rispetto ai controlli [36].

Per conoscere a fondo la neurobiologia dell'autismo, gli studi effettuati negli ultimi venti anni sono di fondamentale importanza, ma c'è ancora tanto da scoprire: infatti, le conoscenze sono ancora puramente descrittive. C'è un urgente bisogno di un modello animale per dare risposta a numerose domande: infatti, a causa della limitata disponibilità di autopsie umane, e dell'eterogeneità della patologia, non si è ancora riusciti a darne una descrizione neurobiologica univoca. Si spera che con i progressi tecnologici e una migliore definizione della genetica della sindrome, sia possibile delineare un profilo più dettagliato della neurochimica e neuroanatomia dell'autismo, per poi conoscerne anche patogenesi e neurobiologia e infine identificare possibili strategie di interventi e trattamenti.

2.2 Disturbi neurodegenerativi

In questo paragrafo, ci si focalizzerà in particolare sui tre disordini neurodegenerativi più studiati dalla comunità scientifica: la demenza senile di tipo Alzheimer, la malattia idiopatica di Parkinson e la malattia di Huntington.

2.2.1 Morbo di Alzheimer

Il morbo di Alzheimer, detta anche demenza senile di tipo Alzheimer, demenza degenerativa di tipo Alzheimer o semplicemente demenza di Alzheimer, è una delle forme più comuni di demenza degenerativa invalidante, che si manifesta in media – tarda età. Questa è stata descritta per la prima volta nel 1906 dal neuropatologo **Alois**

Alzheimer, ma la patogenesi non è ancora del tutto chiara. La malattia è dovuta a una diffusa distruzione di neuroni, principalmente attribuita alla β-amiloide, una proteina che, depositandosi tra i neuroni, agisce come una sorta di collante. La malattia è accompagnata da una forte diminuzione nel cervello di acetilcolina, che, essendo un neurotrasmettitore, è una molecola fondamentale per la comunicazione tra neuroni, e, dunque, per la memoria e ogni altra facoltà intellettiva. La conseguenza di queste modificazioni cerebrali è l'impossibilità per il neurone di trasmettere gli impulsi nervosi, e quindi la morte dello stesso, con conseguente atrofia progressiva del cervello nel suo complesso. Particolarmente colpiti da questo processo patologico sono i neuroni colinergici, specialmente quelli delle aree corticali, sottocorticali e, tra queste ultime, le aree ippocampali. In particolare, l'ippocampo è una struttura encefalica che svolge un ruolo fondamentale nell'apprendimento e nei processi di memorizzazione; per questo la distruzione dei neuroni di queste zone è ritenuta essere la causa principale della perdita di memoria dei malati.

Attraverso un modello epidemiologico, **Brookmeyer et al.** (1998) stimarono il numero di pazienti affetti da questo tipo di demenza: se nel 1998 i casi negli Stati Uniti erano 2.32 milioni, di cui il 68% donne, l'incidenza della patologia tra cinquant'anni potrebbe aumentare in modo considerevole [37]. Lo stesso Brookmeyer nel 2007 stima che nel 2050 circa una persona su ottantacinque sarà affetta da tale morbo [38]: l'Alzheimer diventerà, dunque, una vera e propria emergenza sanitaria. Ciò, unito alla limitatezza delle terapie disponibili ad oggi e alle enormi risorse sociali ed economiche che gravano soprattutto su pazienti e familiari, la rende una delle patologie al mondo a maggiore impatto sociale.

Il decorso della malattia è piuttosto eterogeneo, anche se è possibile individuare delle caratteristiche comuni: se nelle prime fasi il paziente non riesce ad acquisire nuovi ricordi, né riesce a ricordare eventi avvenuti recentemente, con l'avanzare della malattia si manifestano irritabilità, aggressività, sbalzi di umore, difficoltà nel linguaggio e progressive disfunzioni sensoriali. La diagnosi è possibile grazie a test

cognitivi, valutazioni comportamentali e alle nuove tecniche di imaging a risonanza magnetica [39]. Le caratteristiche patologiche della demenza di tipo Alzheimer sono: degenerazione di specifiche cellule neuronali e presenza di placche amiloidi, formazioni extracellulari costituite da una parte centrale in cui si accumula la proteina β-amiloide, e una parte periferica in cui si accumulano detriti neuronali, soprattutto frammenti assonali, e ammassi fibrillari, fasci di filamenti elicoidali nel citoplasma dei neuroni che si dislocano nel nucleo, avvolgendolo (Fig. 3).

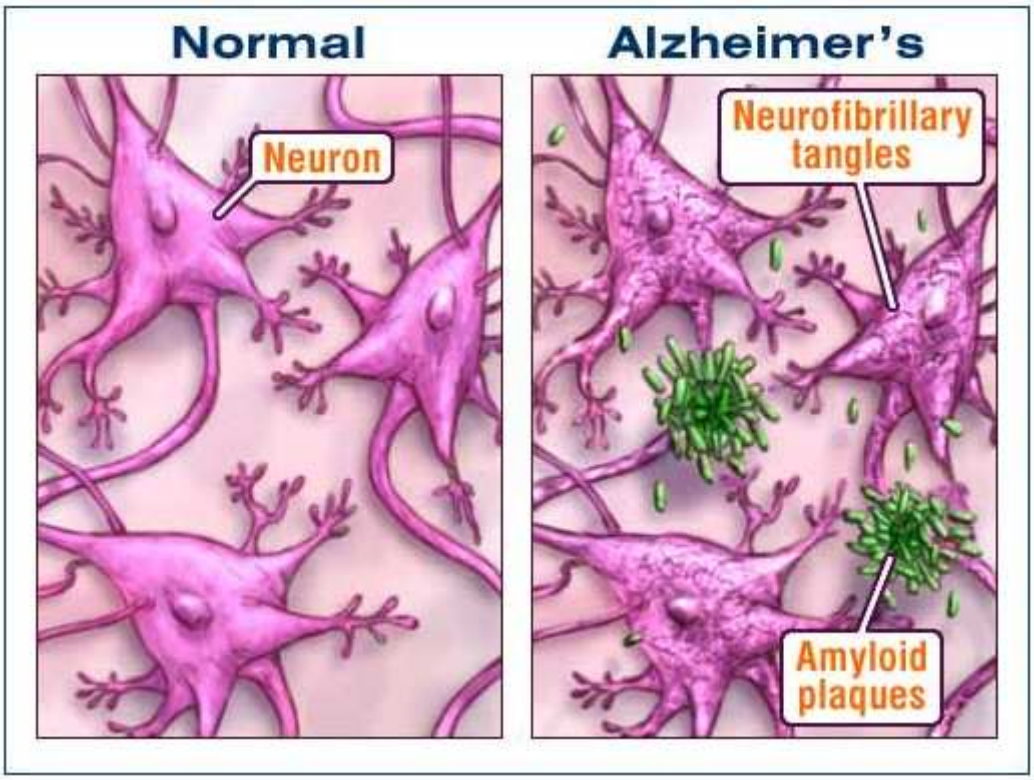


Fig. 3 Rappresentazione di neuroni in individui di controllo e affetti da morbo di Alzheimer.

Per quanto riguarda le alterazioni, le più evidenti si osservano nel sistema colinergico del proencefalo e, in alcuni casi, nel sistema noradrenergico e somatostatinergico che innerva il telencefalo [40].
Macroscopicamente, il cervello dei pazienti affetti dal morbo di Alzheimer presenta un'importante atrofia, con una riduzione del peso di circa il 35% come conseguenza della neurodegenerazione

progressiva di strutture cerebrali limbiche e corticali, in particolar modo nel lobo temporale (**Ricci et al.**, 2009) [41].

Le alterazioni rispetto agli individui di controllo riguardano sia il numero di neuroni, sia la morfologia degli stessi.

La diminuzione del numero di neuroni è attestato in numerosi lavori reperibili in letteratura. Il team di **Whitehouse** (1982) individuò, attraverso studi *post mortem*, una significativa riduzione delle cellule neuronali colinergiche (neuroni che producono acetilcolina come neurotrasmettitore e sono tipici del sistema nervoso parasimpatico) nella corteccia dei pazienti affetti da morbo di Alzheimer. I risultati dello studio mostrano che i neuroni colinergici nella regione di Meynert vanno incontro a un'importante degenerazione, con diminuzione della popolazione neuronale anche del 75%. Il dato più importante di questo studio è la dimostrazione della degenerazione selettiva di questo particolare tipo neuronale [42]. Stesso risultato è stato ottenuto dallo stesso Whitehouse nel 2004. Infatti, comparando i nuclei di Meynert di individui affetti da morbo di Alzheimer e controlli, egli mostrò una sostanziale diminuzione di neuroni. La perdita della popolazione neuronale potrebbe rappresentare una correlazione anatomica di uno squilibrio colinergico nel disturbo [43]. Comparando sezioni del nucleo basale di Meynert, sottoposti a diverse colorazioni istologiche, di due soggetti con morbo di Alzheimer e quattro di controllo, **Saber** (1985) dimostrò una perdita di innervazione colinergica, associata alla perdita di cellule nervose e apparentemente dovuta alla maturazione delle placche neuro fibrillari [44]. Risultati in accordo con le tesi degli studi citati precedentemente sono dati anche dall'analisi condotta da **Arendt et al.** (1983), studiando cinquantotto pazienti con disturbi neuropsichiatrici e quattordici controlli. Il risultato dimostrò una perdita di neuroni nel nucleo basale di Meynert di circa il 70% per quanto riguarda il morbo di Alzheimer [45]. Due anni dopo, lo stesso team, studiando cinque individui patologici, dimostrò una stretta correlazione non lineare tra la perdita di neuroni nel nucleo basale sopracitato e la formazione delle placche neuritiche in varie zone corticali [46]. L'atrofia corticale

e la perdita di cellule nel nucleo basale colinergico è, quindi, una caratteristica del morbo di Alzheimer. In letteratura, è possibile reperire studi più quantitativi che attestano questa peculiarità. Nelle sezioni di nucleo di Meynert spesse 50 μm e preparate con colorazione di Nissl di otto pazienti con tale disturbo e dodici controlli, **Cullen et al.** (1997) mostrarono che la perdita di neuroni è considerevole, fino anche all'89% rispetto ai controlli. Mentre i volumi della sostanza bianca sono inalterati, quelli corticali sono notevolmente ridotti. L'atrofia della sostanza grigia è prominente nella corteccia temporale e frontale degli individui affetti dal disturbo, rispetto ai controlli [47].

La diminuzione del numero di neuroni nel nucleo basale di Meynert potrebbe far pensare che in realtà le cellule siano atrofizzate, e quindi difficili da distinguere dalla neuroglia a causa delle dimensioni: ciò potrebbe avere importanti implicazioni terapeutiche. **Allen et al.** (1988), quindi, utilizzando un colorante neurone-specifico, hanno ottenuto un profilo di distribuzione neuronale attraverso *image analysis*, comparando pazienti e controlli. C'è una perdita di circa il 29% di neuroni; inoltre, c'è una significativa riduzione dell'area neuronale media, dovuto a preponderanza di neuroni piccoli. Ciò suggerisce che i neuroni non sono completamente mancanti, ma sono più piccoli e quindi esclusi dai precedenti studi, in cui si usava una colorazione non neurone-specifica [48].

Per quanto riguarda la morfologia neuronale, in uno dei primi studi quantitativi effettuati su tale disturbo, **Coleman** e il suo team (1987) condussero un'analisi sia su topi che su scimmie anziane e su individui affetti da morbo di Alzheimer. Essi sottolinearono come, nonostante i fattori limitanti di un'analisi condotta su individui affetti dal disturbo, quali ad esempio le differenze tra specie e gli effetti prodotti sul tessuto durante la preparazione dello stesso, ci sono evidenti alterazioni nel numero dei neuroni e nell'estensione dendritica, localizzati in zone specifiche del cervello [49].

Le alterazioni si riscontrano anche in altre aree cerebrali: **Palay et al.** (1989), attraverso uno studio qualitativo e quantitativo dei neuroni del punto blu, situato nel tronco encefalico, evidenziarono delle differenze

negli individui patologici rispetto ai soggetti di controllo. La lunghezza del punto blu, infatti, è minore; inoltre, i neuroni appaiono più gonfi e alterati nella morfologia, con rami dendritici più sottili e radi. Il numero dei neuroni è ridotto, soprattutto nella parte rostrale, rispetto alla centrale e caudale. La presenza di depressione nei soggetti patologici è accompagnata da una perdita più importante di neuroni [50].

Attraverso modelli animali del disturbo, come i topi transgenici APP/PS1 utilizzati dal team di **Knafo et al.** (2009), si sono esaminati i neuroni dell'amigdala, regione cerebrale di cui si è già sottolineato il ruolo cruciale nello sviluppo della patologia. Non c'è perdita di neuroni in questa zona, ma si è riscontrata un'alterazione dell'albero dendritico e un sostanziale decremento del numero di spine dendritiche. Questi cambiamenti probabilmente contribuiscono alla caratterizzazione del morbo di Alzheimer [51].

Oltre a perdita di sinapsi e morte neuronale, sono stati registrati cambiamenti nella morfologia delle cellule gliali, il cui ruolo potrebbe essere fondamentale nella cascata di eventi neurodegenerativi che avvengono nel morbo di Alzheimer (**Hu et al.**, 1998) [52].

In conclusione, sono tanti gli studi reperibili in letteratura che attestano, oltre a una diminuzione della popolazione neuronale, anche alterazioni morfologiche dei neuroni stessi in particolari aree del cervello. Ciò potrebbe avere un ruolo fondamentale per la diagnosi precoce della demenza, ma anche per la progettazione e per l'attuazione di strategie terapeutiche.

2.2.2 Malattia idiopatica di Parkinson

La malattia idiopatica di Parkinson, meglio conosciuta come morbo di Parkinson, fu descritta per la prima volta dal medico **James Parkinson** nel 1817. Rappresenta il disordine più frequente nell'età avanzata, ed è causato da una grande varietà di disturbi, che includono forme neurodegenerative e sintomatiche. Questa malattia è dovuta, infatti, alla degenerazione cronica e progressiva delle strutture nervose

che costituiscono il sistema extrapiramidale. Tale alterazione si estrinseca maggiormente in un'area del sistema nervoso centrale detta *substantia nigra*, un nucleo situato a livello del mesencefalo in cui viene prodotta la dopamina, un neurotrasmettitore in grado di facilitare il movimento del corpo agendo su recettori presenti nel nucleo striato. In definitiva, il morbo di Parkinson è caratterizzato dalla perdita dei gruppi cellulari in grado di *facilitare* il movimento attraverso la secrezione di dopamina. Questi elementi sono la causa scatenante delle caratteristiche cliniche peculiari della patologia, sia motorie:

- Bradicinesia, cioè difficoltà del paziente ad iniziare un movimento, e quando si cerca di effettuarlo, lo stesso risulterà rallentato;
- Tremore;
- Instabilità posturale;
- Rigidità;

che non motorie, dovute alle degenerazione progressiva delle reti neurali, in particolare quella dopaminergica. La perdita di neuroni non è circoscritta alla *substantia nigra*, ma può interessare altre zone cerebrali; ciò a causa del fatto che oltre al morbo di Parkinson propriamente detto, ci sono molte altre patologie, dette parkinsonismi, sia di tipo degenerativo che di altro tipo [53].

Il morbo di Parkinson, in definitiva, coinvolge molti sistemi nervosi ed è il risultato di cambiamenti in specifiche cellule nervose; ad esso è correlata perdita neuronale con successiva proliferazione astrocitaria, in particolar modo nei nuclei del tronco encefalico, nel diencefalo basale e, in grado minore, nella corteccia cerebrale.
Per quanto riguarda l'evoluzione delle alterazioni neuropatologiche correlate al morbo di Parkinson, una stadiazione proposta di recente distingue 6 fasi (**Braak et. al.** 2003). La degenerazione inizia nel tronco encefalico inferiore e nel sistema olfattorio: gli stadi presintomatici I e II mostrano lesioni nel midollo allungato e del bulbo olfattorio, con la preservazione del mesencefalo e degli altri distretti.

Nello stadio III le alterazioni citoscheletriche e la perdita neuronale sono inizialmente lievi, ma si aggravano con il tempo, e si localizzano nel *locus coeruleus*, nei nuclei centrali del bulbo, nei nuclei del diencefalo basale e nel Sistema Nervoso Centrale postero-laterale/postero-mediale. Nello stadio IV vengono colpiti il lobo temporale antero-mediale e la neocorteccia. Infine, negli stadi terminali V e VI, il processo patologico raggiunge la corteccia telencefalica, prima a livello della regione prefrontale e delle aree di sensitive secondarie associative, poi a livello delle aree motorie e sensitive primarie, coinvolgendo quasi l'intera neocorteccia [54]. **Jelliner** nei suoi studi ha rivelato, oltre al coinvolgimento del tronco dell'encefalo, anche lesioni a livello dell'allocorteccia, dell'amigdala e del giro del circolo, corrispondenti allo stadio patologico V-VI [55].

Anche per il morbo di Parkinson sono reperibili in letteratura vari studi, riguardanti conta cellulare e morfologia neuronale, che riguardano diverse regioni cerebrali.

Uno dei primi studi a riguardo risale al 1983, quando il team di **Candy**, combinando studi neuropatologici e neurochimici sul nucleo di Meynert, ha dimostrato che l'attività colinergica è ridotta nei pazienti affetti da morbo di Parkinson, così come accade anche nel morbo di Alzheimer. Ciò è dovuto a una sostanziale anormalità nell'attività istochimica delle cellule neuronali, che porta a una degenerazione degli assoni colinergici. Inoltre, è stato osservata una perdita di neuroni in suddetto nucleo, più consistente che quella osservata per gli individui con morbo di Alzheimer. [56]. **Nakano et al.** (1984) si sono focalizzati sul nucleo basale di Meynert, osservata in undici pazienti con morbo di Parkinson e in tredici soggetti di controllo. Si osservò anche in questa regione una diminuzione del numero dei neuroni [57].

Secondo **Ricci**, i caratteristici movimenti involontari del corpo, la bradicinesia, l'instabilità posturale e la rigidità, tipici della patologia, sono dovuti ad una selettiva perdita di neuroni dopaminergici, cioè i neuroni il cui principale neurotrasmettitore è la dopamina, nella *substantia nigra* [41]. Un altro studio interessante sul tale regione

cerebrale è stato condotto da **Damier et al.** (1992). Essi, per dimostrare la perdita di neuroni contenenti dopamina, utilizzano una immunocolorazione a base di calbindina per isolare la *substantia nigra*. All'interno di essa, sono identificate le parti contenenti i neuroni dopaminergici. Si osserva una perdita neuronale a gradiente caudorostrale, con picchi anche del 98% [58]. **Rinne** (1989), analizzando la *substantia nigra* di dodici pazienti affetti da morbo di Parkinson e diciotto soggetti di controllo, mostrò come la percentuale di neuroni presente in tale regione è ridotta, dal 31% al 49% rispetto ai controlli. Il numero di neuroni nella parte laterale della *substantia nigra* è correlata negativamente alla rigidità e all'ipocinesia, tipici del disturbo. Il grado di demenza degli individui, inoltre, è in significativa correlazione con la diminuzione dei neuroni nella parte mediale della *substantia nigra* [59]

Osservando il tronco encefalico, **Halliday et al.** (1990), eseguendo analisi autoptiche immunoistochimiche, hanno identificato una degenerazione di vari gruppi cellulari neuronali in pazienti con morbo di Parkinson, nel *nucleus raphe* del tronco e nel nucleo vagale dorsale [60].

Per quanto riguarda la morfologia, uno studio interessante è stato condotto, parallelamente su individui colpiti da Alzheimer e Parkinson, nel 1989 dal team di **Palay**. Analizzando il *locus coeruleus*, infatti, si notò che negli individui affetti da Parkinson la lunghezza rostro caudale del *locus coeruleus* è minore che negli individui affetti da Alzheimer. La morfologia neuronale, inoltre, è maggiormente alterata: i corpi cellulari risultano più gonfi, e i dendriti sono più corti e sottili, con arborizzazione ridotta o assente. Il nucleo dei neuroni è di dimensioni minori, sia rispetto ai controlli sia rispetto ai malati di Alzheimer. Anche in questo caso, se in concomitanza con il morbo di Parkinson è presente depressione, la perdita neuronale è accentuata [50].

Nel 2007, **Solis et al.** studiarono le alterazioni morfologiche dei dendriti nei neuroni piramidali della corteccia prefrontale e i neuroni del *putamen* e *del nucleus accumbens* in un modello murino della

patologia. I risultati mostrarono una diminuzione della densità delle spine dendritiche senza significativi cambiamenti nella lunghezza dendritica o nell'arborizzazione [61].

2.2.3 Malattia di Huntington

La malattia di Huntington, conosciuta anche come Còrea di Huntington, è una malattia degenerativa del sistema extrapiramidale. Il nome deriva dal suo scopritore, **George Huntington**, che per primo la descrisse nel 1872 [62]. Le cause della malattia sono da ricercare nella degenerazione geneticamente programmata dei neuroni in alcune particolari aree cerebrali. Ciò provoca movimenti del corpo che il paziente non riesce a controllare, perdita delle facoltà intellettive e disturbi emozionali.

E' una malattia ereditaria ed è caratterizzata da una mutazione di un particolare gene. Se la mutazione del gene è assente, il neonato non può contrarre la malattia né può trasmetterla alla sue prole; al contrario, il decorso è inevitabile, il cui esordio è in genere tra i quaranta e i cinquant'anni. E' un disturbo annoverato tra le malattie rare: secondo la *Prima Associazione Italiana della Corea di Huntington*, infatti, la prevalenza è di circa tre – sette casi ogni centomila abitanti con discendenza europea occidentale, e uno su un milione di abitanti con discendenza asiatica. La trasmissione della patologia è di tipo autosomico dominante a penetranza completa, ciò vuol dire che un bambino con un genitore affetto dalla malattia di Huntington ha il 50% di probabilità di contrarre tale disturbo, salvo complicanze dovute al numero di ripetizioni della sequenza nucleotidica interessata.

I primi sintomi della malattia sono stati depressivi, alterazione della personalità ed irrequietezza, seguita da una compromissione dei sistemi motori: sono caratteristici, infatti, i movimenti involontari dei muscoli facciali e degli arti, dapprima brevi e distali, poi sempre più frequenti e duraturi, tanto da dare luogo a una strana danza (da qui il nome còrea, dal greco "danza"). A causa di questi spasmi, anche la fonazione e i movimenti oculari sono compromessi. Negli ultimi stadi

della malattia, l'irritabilità sfocia in vere e proprie turbe della psiche, caratterizzate da idee deliranti a carattere persecutorio fino a stati demenziali conclamati. La durata della malattia è di circa quindici – venticinque anni, e il decesso avviene per varie cause intercorrenti, in particolare complicazioni polmonari.

La malattia di Huntington è provocata da una mutazione di una proteina molto grande, la huntingtina (Fig. 4), ampiamente diffusa in tutto il corpo, una circostanza che rende difficile studiare i meccanismi d'azione della proteina mutante. Nono è ancora chiaro però come la proteina mutante causi la perdita di cellule cerebrali nei pazienti colpiti da questa malattia [63][64].

Sono reperibili in letteratura interessanti evidenze di perdita e alterazione della morfologia neuronale nella malattia di Huntington in varie regioni cerebrali.

Hedreen et al. (1991) hanno effettuato la conta neuronale *post mortem* nella corteccia cerebrale di cinque individui patologici e cinque di controllo. La diminuzione del numero di neuroni nei soggetti affetti dalla malattia è significativa soprattutto nel sesto strato della corteccia (57%). Anche nel quinto strato il numero dei neuroni è significativamente ridotto (anche del 71% rispetto ai controlli) [65].

Attraverso uno studio morfologico *post mortem* riguardante non solo la malattia di Huntington, ma anche la schizofrenia, **Rajkowska et al.** (1998) studiarono le anormalità delle cellule neuronali nella corteccia cerebrale. Per comprovare queste alterazioni, si sono studiate cervelli autoptici di nove pazienti con schizofrenia, dieci pazienti di controllo e sette pazienti con malattia di Huntington. Si è usato un analizzatore di immagini tridimensionale per misurare i perimetri di cellule neuronali e gliali nelle aree di Brodmann 9 e 17. Nelle corteccie dei pazienti con malattia di Huntington, la degenerazione neuronale è evidente, come pure la diminuzione della dimensione neuronale e della densità dei neuroni di grandi dimensioni. Di contro, c'è un drammatico aumento della densità delle cellule gliali [66].

Il team di **Richfield**, invece, ha mostrato una perdita dei neuroni nei gangli basali nella malattia di Huntington [67].

Slow et al. (2003), in virtù del fatto che la malattia di Huntington è un disturbo prevalente motorio, prima che cognitivo, si focalizzarono sui neuroni nel corpo striato, responsabili del movimento. Utilizzando un modello murino della patologia, mostrarono come i deficit motori siano altamente correlati alla perdita di tali neuroni [68]. Sempre per quanto riguarda il corpo striato, **Klapstein et al.** (2001) utilizzando un modello animale della patologia mostrarono nei neuroni non solo alterazioni elettrofisiologiche, quali l'aumento della resistenza di ingresso della membrana o alterazione nei potenziali d'azione, ma anche alterazioni morfologiche, tra cui la diminuzione della densità delle spine dendritiche, diametri minori dei soma cellulari e aree minori. Ciò si ripercuote sulla circuiteria di comunicazione con i gangli della base [69].

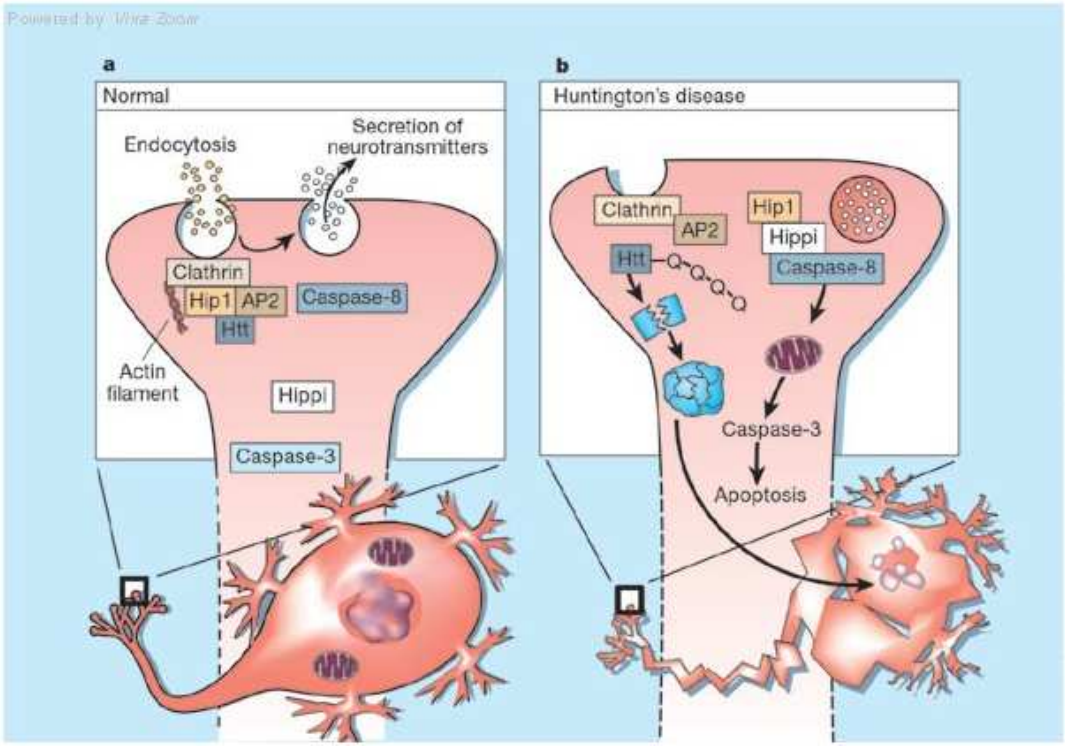


Fig. 4 Cambiamenti introdotti dalla mutazione della proteina detta huntingtina

Nel 2004, **Baquet** studio un fattore neurotrofico che modula differenziazione e funzione sinaptica. Riducendo l'espressione di tale fattore nella corteccia cerebrale nei topi, si induce la mutazione del

gene responsabile della malattia di Huntington. Rispetto ai controlli, il modello animale della patologia presenta un corpo striato di volume inferiore e alterazioni morfologiche dei neuroni in esso presenti: soma cellulari atrofici, con dendriti più sottili, e una densità ridotta di spine dendritiche. Ciò vaglierebbe l'ipotesi di un collegamento tra il fattore neurotrofico corticale e la morfologia neuronale nel corpo striato [70]. Attraverso un modello murino generato *ad hoc* per la riproduzione delle caratteristiche salienti della patologia, **Lerner et al.** (2012) hanno dimostrato la perdita di neuroni. Analisi stereologiche effettuate su tali topi a quattro mesi di vita non rilevarono mutazioni nel volume delle proiezioni del corpo striato, ma fu individuato al dodicesimo mese di vita atrofia sia nei maschi che nelle femmine. Analisi del corpo calloso nei maschi rilevarono la perdita dei neuroni intorno al ventesimo – ventiseiesimo mese, come anche una perdita di spine dendritiche nei neuroni del corpo striato, soprattutto nei rami dendritici distali; si evidenziò anche una minore complessità della struttura dendritica, come rivelato dall'analisi di Sholl [71].
Infine, **Spires et al.** (2004), evidenziarono per la prima volta anormalità dendritiche in un modello murino della patologia, simulante la neuro degenerazione tipica degli individui affetti da còrea [72].

Essendo malattie neurodegenerative, importante sviluppo futuro potrebbe essere una comprensione di come la perdita di neuroni e le alterazioni della morfologia si sviluppino nel tempo. La maggior parte degli studi citati, infatti, analizzano cervelli autoptici: è quindi un'analisi statica.

2.3 Alterazioni morfologiche dei neuroni in altre patologie

Sono reperibili in letteratura numerosi studi che attestano cambiamenti nella morfologia neuronale a seguito di alterazioni dello stato fisiologico.

Le ricerche di **Martinez-Tellez et al.** (2005) sono state condotte modelli animali del diabete mellito indotto da streptozotocina, molecola che, danneggiando selettivamente le cellule β-pancreatiche, causa l'insorgenza della patologia. Il team ha analizzato la morfologia dei neuroni piramidali della corteccia prefrontale, della corteccia occipitale e dell'ippocampo per studiare i cambiamenti prodotti dall'innalzamento della glicemia in cavie da laboratorio. A queste, tenute a un certo livello di glucosio e sacrificate dopo due mesi, è stato estratto il cervello e, previa colorazione al Golgi, è stata effettuata l'analisi di Sholl sui neuroni corticali ottenuti. Nelle cavie con diabete mellito,la lunghezza dei dendriti nelle cellule piramidali nelle regioni considerate diminuisce dal 20% al 45% rispetto ai controlli. Inoltre, diminuisce dal 36% al 58% anche la densità delle spine dendritiche: ciò è evidente per le cellule piramidali dell'ippocampo. L'analisi di Sholl mostra che nei topi diabetici il numero di intersezioni è ridotto rispetto ai soggetti di controllo. Questo risultato dà prova di come il diabete mellito influenzi la morfologia dendritica in strutture implicate nei processi cognitivi, quali corteccia pre-frontale e occipitale e ippocampo [73].

Riley et al. (1978) hanno effettuato esperimenti per comprovare i danni sulle strutture nervose dovuti all'assunzione di alcool. Nello studio si sono utilizzati topi, sottoposti a una dieta contenente alcool per quattro mesi. Il consumo a lungo termine di alcool provoca una diminuzione significativa nel numero di spine dendritiche nelle cellule piramidali dell'ippocampo, producendo danni morfologici al sistema nervoso centrale [74].

Anche l'ipotiroidismo può provocare danni alla normale struttura neuronale: **Ruiz-Marcus et al.** (1980) hanno dimostrato che i neuroni piramidali nella corteccia visiva di ratti ipotiroidei presentano cambiamenti nel numero e nella distribuzione delle spine dendritiche. E' un'alterazione, però, reversibile: somministrando ormoni tiroidei agli stessi, però, le cellule piramidali ritornano alla struttura fisiologica [75].

Tramite colorazione al Golgi sono stati valutati i cambiamenti morfologici di neuroni che presentano alterazione dell'idrolisi lisosomiale su modelli animali. **Walkley et al.** (1988) , infatti, hanno mostrato come neuroni piramidali corticali presentino ingrossamento dell'assone. Le strutture terminali degli stessi, inoltre, appaiono più evidenti, e spesso più lunghe rispetto ai neuriti osservati in altri studi.
Assunzione ripetuta di droghe psicostimolanti, quali cocaina o anfetamine, producono cambiamenti significativi a livello cerebrale. E' importante conoscere a fondo questo tipo di alterazioni, perché si pensa che da essi dipenda lo sviluppo della dipendenza [76]. **Robinson et al.** (1999), iniettando cocaina o anfetamina in ratti, successivamente sacrificati, e sui cervelli dei quali è stata effettuata la colorazione Golgi-Cox, dimostrarono che il numero di ramificazioni dendritiche e la densità delle spine dendritiche nel *nucleus occumbens*, e nelle cellule piramidali del quinto strato della corteccia prefrontale. Ciò fa pensare che le conseguenze neuro comportamentali all'assunzione di droghe psicostimolanti sia dovuta alla necessità di riorganizzare le connessioni sinaitiche [77]

In conclusione, la revisione della letteratura dimostra come numerosi studi sono stati effettuati sia su uomo sia in modelli animali, in particolare murini, per analizzare le anormalità del sistema nervoso collegate a varie patologie. In particolare, frequentemente tali alterazioni riguardano sia la morfologia dei neuroni, sia la loro numerosità. E' evidente, quindi, quanto sia di fondamentale importanza l'utilizzo di strumenti informatici che permettano di quantificare l'entità delle alterazioni presenti a livello neuronale.

Capitolo 3

Tecniche software

In questo capitolo verranno descritte tutte le tecniche implementate ed utilizzate durante il lavoro di tesi.

In primo luogo, si descriverà Ne.Mo., il software per l'image processing e l'analisi morfologica e topologica dei neuroni. Lo scopo di questi algoritmi è quello di estrarre il maggior numero possibile di informazioni sulla morfologia, sulla topologia e sull'organizzazione delle cellule, a partire da immagini, ottenute al microscopio, che le ritraggono. L'importanza dello studio metrico della morfologia delle cellule neuronali deriva dal fatto che in varie patologie, in particolare nell'autismo, esse subiscono una forte alterazione, come si è più volte sottolineato nel Capitolo 2: soltanto dopo aver compreso e misurato in maniera precisa e ripetibile tutti i dettagli, sarà possibile chiarire quali siano i danni a cui esse vanno incontro. In particolare, ci si soffermerà sulla 3-way PCA, poiché se ne sono implementati gli algoritmi in linguaggio di programmazione Java in una seconda fase del lavoro.

Gli algoritmi, implementati in Matlab, sono stati scritti in parte durante il mio lavoro di tesi triennale [78], in parte dall'Ing. Lucia Billeci nel suo lavoro di tesi specialistica [79].

Durante questo lavoro di tesi si è effettuato un *debug* del software Ne.Mo.; inoltre, si è cercato di rendere il software il più possibile user-friendly, essendo esso destinato a laboratori biologici, in cui, quindi, non è garantita la presenza di operatori con buone conoscenze informatiche. Una volta controllata la corretta funzionalità delle varie

interfacce grafiche, con i relativi editor che lo compongono, si è proceduto con la compilazione del codice, in modo da ottenere un file eseguibile. In questo modo, Ne.Mo. è utilizzabile anche su computer sprovvisto della licenza MathWorks.

Nella seconda fase del lavoro di tesi, è stato implementato, in linguaggio di programmazione Java, un plug-in che effettua la 3way-PCA. Questa è tecnica di analisi multivariata, implementata da **Leardi** e collaboratori in linguaggio Matlab e utilizzata anche in Ne.Mo.: infatti, permette di analizzare e confrontare ulteriormente parte delle variabili estratte dall'analisi delle cellule neuronali. La scelta del linguaggio di programmazione è stata dettata dal fatto che ImageJ, uno dei software per l'*image processing* e *analysis open-source* più utilizzato nei laboratori, è scritto in linguaggio Java. Come già sottolineato nel Capitolo 1, Image J dà la possibilità non solo di utilizzare i plug-in di altri utenti, ma anche di caricarne di nuovi: si è pensato, dunque, di condividere il plug-in che effettua la 3way-PCA, essendo questa una tecnica capace di analizzare i dati nelle loro correlazioni e quindi di giungere a conclusioni complete ed esaustive.

Questo algoritmo verrà riportato integralmente nell'Appendice A.

Nella prima parte del capitolo si descriverà Ne.Mo. in ogni sua parte, sottolineando le differenze apportate rispetto alle versioni presentate negli scorsi lavori di tesi. Si passerà quindi ad una descrizione dettagliata del plug-in implementato in linguaggio di programmazione Java. Per ogni tecnica utilizzata la cui comprensione richiede qualche conoscenza teorica, verrà riportata brevemente la teoria a cui si fa riferimento.

I risultati ottenuti dall'applicazione delle tecniche qui descritte verranno riportati e discussi nel Capitolo 4.

3.1 Ne.Mo.

Ne.Mo., acronimo di "NEuron MOrphological tool", è un software open-source per il *processing* e l'analisi di cellule, in particolare neuroni. Infatti, è un software pensato per analizzare cellule neuronali: queste sono, infatti, coinvolte in alterazioni sia nel numero che nella

morfologia in molte patologie, ad esempio nell'autismo, come già più volte sottolineato precedentemente.

L'obiettivo principale del software è quello di essere il più possibile user-friendly, in modo da poter essere usato anche da utenti che non conoscono Matlab o altri linguaggi di programmazione. Nonostante la sua semplicità, Ne.Mo. è un valido e accurato strumento per analisi sia statiche che dinamiche.

Con questo tool, è possibile ottenere informazioni sulla morfologia cellulare a partire da immagini, ottenute al microscopio, rappresentanti singoli neuroni, reti neurali o slice.

L'interfaccia grafica iniziale si presenta come in Fig.1: questa ci permette di accedere ai vari editor e GUI implementati nei precedenti lavori di tesi. Ciascun input del menù a tendina sulla barra dei menù è creato attraverso appositi comandi Matlab e ad ognuno di questi è associata una funzione, che rimanda a comandi scritti nello stesso editor o in editor differenti.

Prima di caricare le immagini da elaborare, è importante nominarle con particolare attenzione: il software, infatti, estrae automaticamente dal nome le informazioni sulla cellula fotografata. Nel nome di ogni singola immagine, è importante che si faccia riferimento a:

- *Tipologia dell'immagine*: "p" per "*photography*", "b" per "*binary*" e "s" per "*skeleton*". Gli algoritmi per l'elaborazione salvano automaticamente l'immagine binaria e lo scheletro del neurone seguendo questo schema;
- *Numero della coltura*: nel caso in cui si seguano contemporaneamente più colture, esse si distinguono assegnando a ciascuna un numero identificativo;
- *Numero della cellula*: ogni cellula di una stessa coltura è identificata da un numero, assegnato dall'operatore;
- *Giorno in cui la cellula è stata fotografata*: la numerazione parte dal giorno in cui la cellula viene fotografata per la prima volta;
- *Età della cellula in giorni*: la numerazione parte dal primo giorno di vita della cellula in coltura.

Riguardo alle prima due, si inseriscono entrambe le informazioni nel nome perché spesso non si fotografano le cellule dal primo giorno della messa in coltura.
In definitiva, il nome dell'immagine sarà del tipo:

Tipologia dell'immagine_Numero della coltura_Numero della cellula_Giorno in cui la cellula è stata fotografata_Età della cellula in giorni.formato dell'immagine

Il caricamento è possibile con l'opzione "Open" presente in "File": in questo modo, si apre la finestra standard di ricerca dei file, e può essere selezionata un qualsiasi file presente nell'hard disk del computer. Selezionata la directory e, infine, l'immagine desiderata, viene visualizzata l'immagine in Ne.Mo., ed è possibile avviare l'elaborazione.

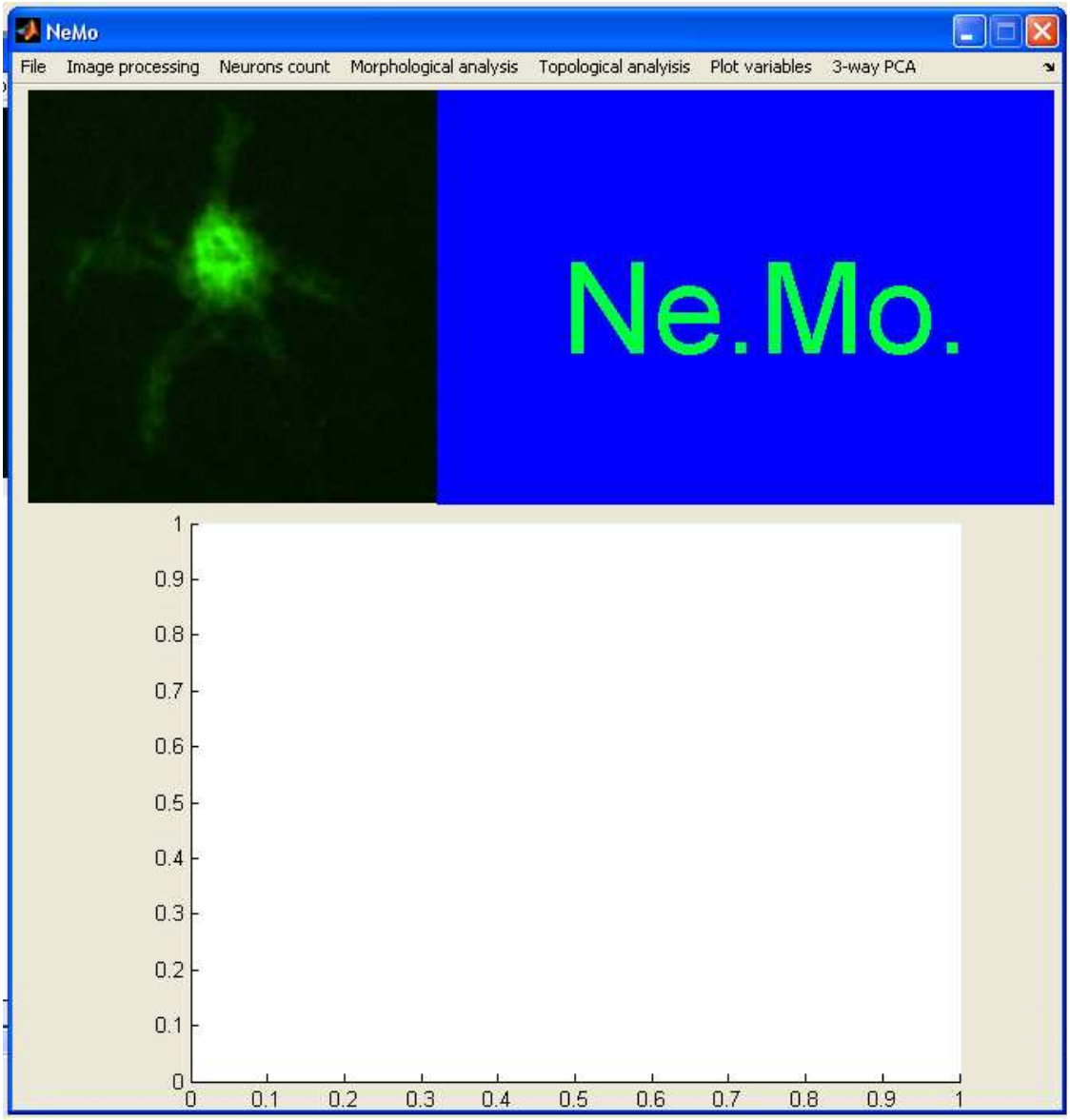


Fig. 1 Interfaccia grafica iniziale di Ne.Mo.

A questo punto, l'utente può:

- Eseguire l'*image processing* dell'immagine: è possibile avviare un'elaborazione semi-automatica o un'elaborazione con maggiore libertà nella scelta dei parametri;
- Eseguire la conta cellulare;
- Eseguire l'analisi morfologica, una volta ottenuto lo scheletro del neurone attraverso *image processing*;
- Eseguire l'analisi topologica di immagini di slice rappresentanti più neuroni;
- Plottare le variabili morfologiche ottenute al punto precedente;

- Eseguire la 3way-PCA.

Si descriveranno sinteticamente i vari punti.

3.1.2 Image Processing

Per ottenere lo scheletro dell'immagine digitale caricata in Ne.Mo., è possibile, come già più volte rimarcato, avviare un'elaborazione semi-automatica, implementata attraverso un editor Matlab (Fig. 2), o una con maggiore libertà nella scelta dei parametri e nella sequenzialità dei comandi da eseguire, implementata, invece, attraverso un'interfaccia grafica (Fig. 3)

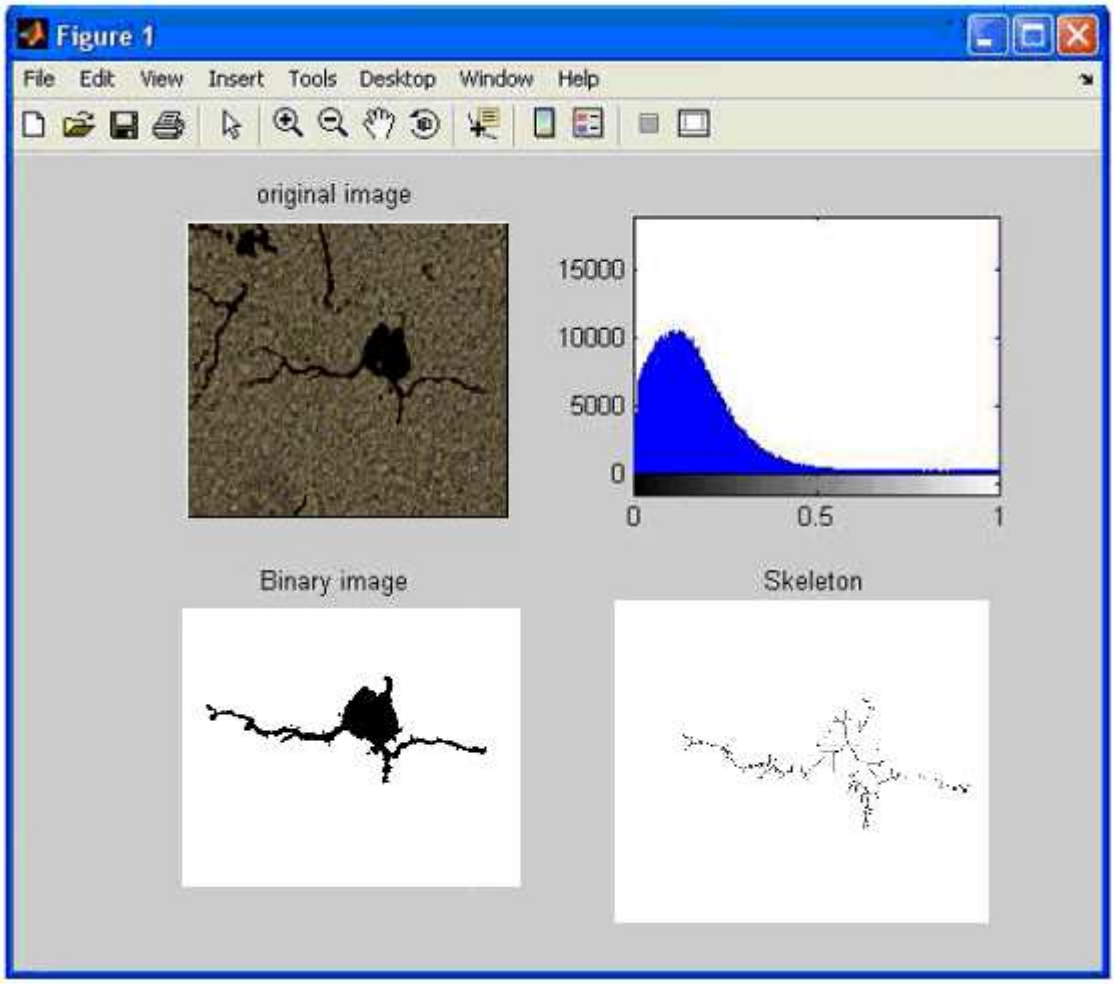


Fig. 2 Editor per l'image processing automatico

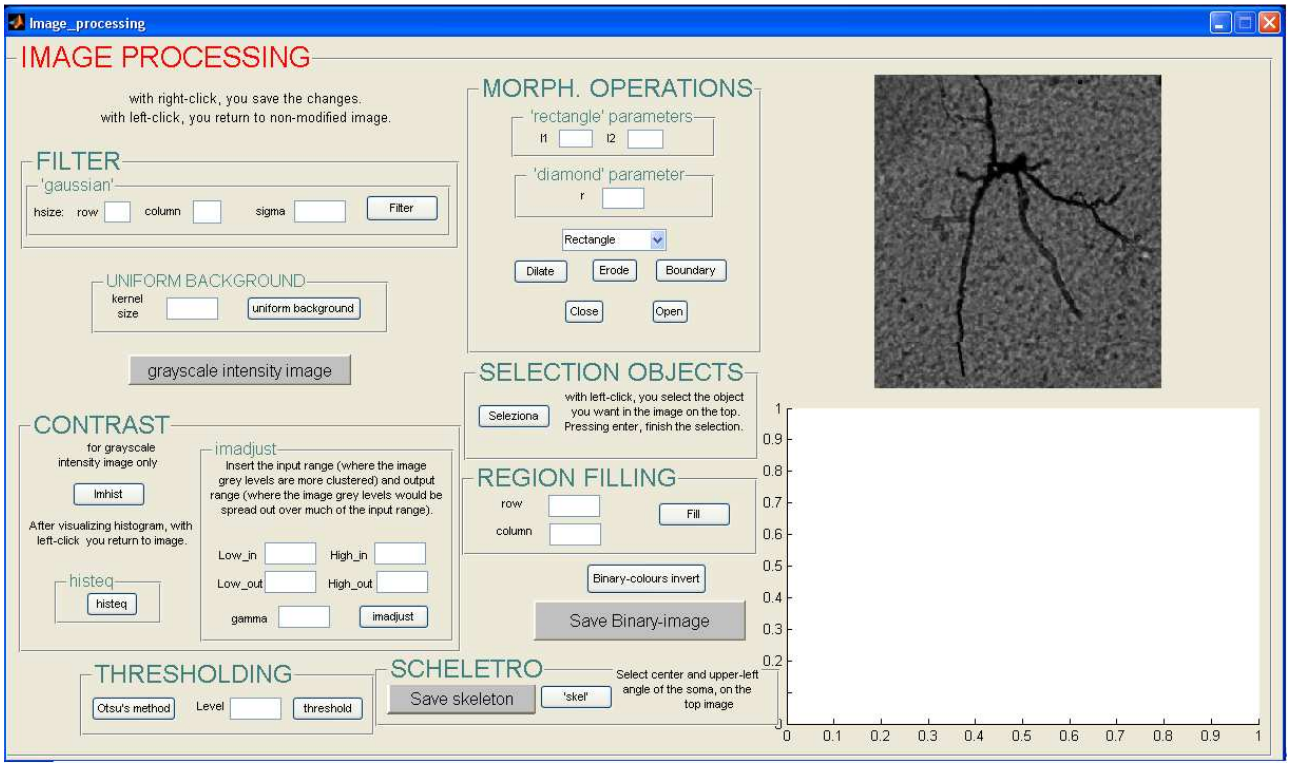


Fig. 3 GUI per l'elaborazione delle immagini digitali

Il funzionamento di tali algoritmi è stato già affrontato e discusso in modo esaustivo in precedenti lavori, quindi per ulteriori spiegazioni si rimanda al punto [78] della bibliografia. Si elencheranno, tuttavia, nella tabella di Fig. 4 tutte le funzioni eseguibili durante l'Image Processing, accompagnate da una breve discussione.

Parametro	Descrizione
Filter	filtra l'immagine attraverso un filtro passa-alto di tipo gaussiano lineare
Uniform Background	elimina le impurità presenti all'interno dell'immagine ed evidenzia la cellule rispetto allo sfondo, il quale si presenterà più compatto ed omogeneo

Greyscale Intensity Image	l'immagine viene convertita da RGB a livelli di grigio, migliorandone il contrasto
Contrast	Migliora il contrasto dell'immagine
Thresholding	Segmentazione dell'immagine al fine di ottenere una fotografia binaria
Morphological Operations	Operazioni morfologiche per migliorare l'immagine (unire dendriti che con operazioni precedenti risultano staccati dal soma, o staccare oggetti che non sono utili al fine dell'analisi)o ottenerne i contorni: è possibile usare *kernel* di due forme: rettangolare o romboidale.
Selection objects	si utilizza per selezionare gli oggetti dell'immagine interessanti dal punto di vista dell'analisi, che vengono selezionati con un click del mouse; il termine della selezione è stabilita dall'utente con la pressione del tasto "Invio"

Region Filling	riempie i contorni della cellula; viene utilizzato quando non si riesce ad ottenere un immagine binaria perfettamente definita
Binary-Colour Invert	inverte i colori dell'immagine
Save Binary Image	Salva l'immagine binaria (neurone nero su sfondo bianco) automaticamente nella stessa cartella dell'immagine di partenza
Scheletro	Ricava lo scheletro dall'immagine binaria

Fig. 4 Tabella che riassume i principali algoritmi per l'image processing implementati in Ne.Mo.

Le migliorie apportate a questi algoritmi hanno fondamentalmente un unico scopo: rendere il programma il più possibile user-friendly
Quindi, per quanto riguarda l'editor per *image processing* semi-automatico, utilizzabile con immagini con poco rumore e ben risolute, si sono utilizzati specifici comandi Matlab che permettono di:

- Interrompere il flusso dell'editor. Infatti, è necessario inserire, durante l'elaborazione, dei parametri. Appariranno automaticamente delle finestre che aiutano l'utente a comprendere il significato di tali parametri, per rendere più agevole la scelta del valore da indicare negli appositi *edit*.
- Alla fine dell'elaborazione, si apriranno automaticamente due finestre, rappresentanti rispettivamente immagine binaria e

scheletro del neurone. Infine, comparirà una terza finestra, che chiederà all'utente se è soddisfatto dell'elaborazione. Nel caso di risposta positiva, le immagini verranno automaticamente salvate nella stessa directory della fotografia di partenza, e l'editor verrà automaticamente chiuso, tornando alla GUI iniziale di Ne.Mo.; in caso contrario, si aprirà in modo automatico la GUI che permette di eseguire gli stessi algoritmi più volte e con diversa sequenza, ma anche di avere più libertà nella scelta dei parametri.

Per quanto riguarda la GUI, si sono aggiunti opportuni comandi Matlab con lo scopo di rendere il software più agevole per l'utente la comprensione dei passaggi da effettuare:

- All'interno della GUI stessa, quando necessario, c'è una breve descrizione di quello che l'utente deve compiere per utilizzare nel modo corretto quel determinato algoritmo: ad esempio, come mostrato in Fig. 5, si fa presente che, per selezionare gli oggetti che si considerano interessanti dal punto di vista dell'analisi da effettuare, l'utente deve selezionarli con un click sinistro del mouse, mentre per interrompere la selezione basta cliccare su "Invio";

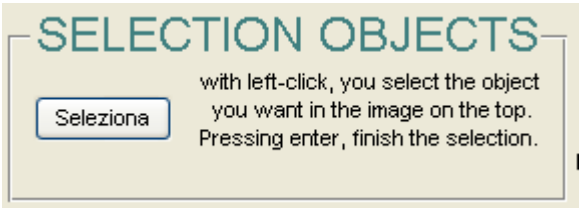


Fig. 5 Esempio di descrizione delle funzionalità di pannello nella GUI

- Soffermandosi con il cursore del mouse sugli edit preposti all'inserimento di determinati parametri, in alcuni casi necessari al corretto funzionamento delle *callback*, si ottengono informazioni sulla funzione delle varie parti che compongono la GUI e sugli eventuali parametri di default, che vengono usati dall'algoritmo qualora l'utente non inserisca tali valori.

Tutto ciò è fatto, come più volte ribadito, per migliorare la comprensione del programma ad un utente non necessariamente esperto nel campo informatico.

3.1.2 Neurons count

Per quanto riguarda la conta neuronale, si è più volte ribadito che questa è utile in molti casi in cui risulta necessario conoscere il numero di cellule presenti in un'immagine.

Le immagini delle *slice* sulle quali è possibile applicare l'algoritmo sono organizzate e nominate con lo stesso criterio descritto per le immagini che ritraggono singoli neuroni.

Rispetto all'algoritmo presentato nel precedente lavoro di tesi, in cui era possibile eseguire la conta solo di cellule neuronali fluorescenti, si è aggiunto un ulteriore editor per la conta di cellule non fluorescenti. Questo è pressappoco lo stesso del precedente, ma tiene conto che le prime sono più chiare rispetto allo sfondo, al contrario delle seconde: nell'algoritmo per la conta di cellule non fluorescenti è necessario prima invertire l'immagine binaria, e poi eseguire le operazioni morfologiche implementate anche nell'editor descritto nel punto [78] della bibliografia.

Entrambi gli algoritmi lavorano in ricorsione: una volta avvenuta la selezione, da parte dell'utente, di un'immagine situata in una certa cartella, l'editor analizza ed effettua la conta cellulare automaticamente su tutte le fotografie presenti in quella cartella. In questo modo, si valuta il numero di neuroni per una particolare *slice* in ogni giorno in cui è stata fotografata.

A fine elaborazione, comparirà una finestra in cui viene riportato il numero di oggetti presenti nell'immagine. Inoltre, è creata automaticamente una cartella, chiamata "Count" nella directory destinata alla *slice* che si sta analizzando. All'interno di questa, vengono salvati, in formato .mat, i vettori contenenti il numero di neuroni per ogni giorno di coltura. Il vettore verrà automaticamente salvato con il seguente nome:

numberOfNeuronsFluo_Numero organo_ Numero slice_Giorno in cui è stata effettuata l'ultima fotografia.mat

Per "Numero organo" si intende, ad esempio, il numero identificativo del cervelletto da cui si sono ottenute le *slice*.

A questo punto, è stata effettuata un'altra aggiunta rispetto al precedente lavoro di tesi. C'è la possibilità, infatti, di ottenere dei grafici che illustrino l'andamento della conta cellulare. In particolare, sono stati pensati due tipi diversi di plot:

- *Plot separate*: si grafica il numero di neurone in funzione dei giorni. L'utente seleziona il file .mat da analizzare e il grafico apparirà automaticamente;
- *Plot Unique*: attraverso una multiselezione, l'utente sceglie i file .mat che vuole visualizzare in un unico plot. Automaticamente, si otterrà il grafico del numero di neuroni per ogni slice selezionata in quella determinata coltura.

3.1.3 Analisi Morfologica

Ottenuto lo scheletro del neurone, si può passare all'analisi morfologica. Lo studio metrico delle cellule neuronali deriva dal fatto che in molti disturbi, come l'autismo, esse subiscono una forte alterazione.

Per quanto riguarda la GUI che permette l'analisi morfologica, questa è stata implementata dall'Ing. Lucia Billeci nel suo lavoro di tesi specialistica: per informazioni più dettagliate ed esaustive delle tecniche utilizzate, si rimanda al punto [79] della bibliografia. In questa sede, si riporta una tabella riassuntiva (Fig. 6) delle operazioni che la GUI effettua.

Parametro	**Descrizione**
Metodo di Sholl Lineare	Plot del numero di intersezioni per area del cerchio in funzione del

	raggio
Metodo di Sholl semi-Log	Plot del logaritmo del numero di intersezioni per area del cerchio in funzione del raggio
Metodo di Sholl Log-Log	Plot del logaritmo del numero di intersezioni per area del cerchio in funzione del logaritmo del raggio
Calcolo centro	Coordinate del centro dello scheletro della cellula
Calcolo raggio minimo	Raggio minimo dal quale si inizia a costruire i cerchi per l'analisi di Sholl
Calcola raggio massimo	Raggio dell'ultimo cerchio da costruire per l'analisi di Sholl
rc	Raggio critico per il quale si ha il maggior numero di intersezioni
Nm	Numero massimo di intersezioni
Schoenen	Rapporto tra il numero massimo di

	intersezioni e il numero di dendriti primari, cioè quelli che hanno origine direttamente dal soma cellulare
k	Coefficiente di regressione di Sholl, ovvero la pendenza della retta che fitta i dati
R	Illustra i coefficienti di correlazione tra i dati
Delta	Identifica il rapporto di determinazione; permette di valutare quale, tra il metodo Semi-Log e il metodo Log-Log è il migliore
Percorso minimo	Minimo percorso tra il centro della cellula e la fine del dendrite più distante
Vettori delle lunghezze minime	distanza tra l'estremità e tutte le intersezioni del cerchio precedente, viene presa la minima; l'intersezione per cui si ottiene la minima distanza diventa la nuova estremità e viene eseguito lo stesso procedimento fino a che non si trova l'intersezione di raggio minimo

Angoli tra vettori	Angoli compresi tra i vettori delle lunghezze minime
Estensione radiale della cellula	Dimensione della cellula, compresi tutti i dendriti
Dimensioni del soma	Il soma viene approssimato a un'ellisse e ne viene calcolata l'area
Area del cono	un algoritmo calcola l'area compreso tra due direttrici che hanno origine nel soma della cellula e tangenti ad essa

Fig. 6 Tabella riassuntiva delle operazioni che si effettuano durante l'analisi morfologica

Ogni volta che viene eseguita una callback, le variabili calcolate vengono inserite in una specifica posizione del datamatrix.
Anche questa GUI lavora in modo ricorsivo: in questo modo, una volta selezionata una cellula in una determinata cartella, si avvia, una per volta, l'analisi per tutte le immagini contenute nella stessa; a fine elaborazione, si avrà, quindi, un datamatrix di dimensioni

*1*177*giorno in cui è stata effettuata l'ultima fotografia*

Tutti i datamatrix vengono salvati in due diverse cartelle, che si creano in modo automatico ad analisi ultimata: la prima, chiamata "*Datamatrix+numero coltura*", nella directory in cui sono presenti le cartelle relative ad ogni coltura; la seconda, chiamata "*Datamatrix*" nella stessa directory in cui è salvato il .exe per l'avvio di Ne.Mo., in

cui vengono salvate tutte le matrici, organizzate per numero di coltura. La seconda cartella facilita, come si spiegherà nei prossimi paragrafi, la costruzione della matrice di dati globale, necessaria per effettuare l'analisi morfologica e, in un secondo momento, la 3way-PCA.
In entrambi i casi, comunque, il datamatrix viene automaticamente salvato con il seguente nome:

DM_Numero della coltura_Numero della cellula_Ultimo giorno in cui la cellula è stata fotografata.mat

Nel caso in cui il giorno della messa in coltura è diverso dal primo giorno in cui le cellule iniziano ad essere fotografate, o nel caso in cui in un particolare giorno le cellule non sono state fotografate, il datamatrix conterrà dei NaN (Not-A-Number) in corrispondenza di questi particolari giorni.

3.1.4 Analisi topologica

Questo tipo di analisi, implementata dall'Ing. Lucia Billeci, è importante nello studio della migrazione delle cellule nervose. Come sottolineato nel Capitolo 2, infatti, studi dimostrano come durante lo sviluppo cerebellare ci sia un'alterazione rispetto ai controlli nella migrazione neuronale in alcune patologie, come ad esempio nei disturbi dello spettro autistico.
Anche in questo caso, le slice sono nominate ed organizzate come descritto nei precedenti paragrafi, e gli algoritmi lavorano in modo ricorsivo.
Nella GUI, dopo un pre-*processing* in cui l'immagine della slice è resa binaria, si ottengono in modo automatico delle informazioni sulla topologia neuronale, riassunte nella tabella di Fig. 7.

Parametro	Descrizione
Intensità media della	Misura della capacità del neurone di esprimere il GFP. Misura utile

fluorescenza	perché l'espressione è funzione del tempo.
Distanza	Distanza tra due layer di cellule
Lunghezza	Si ottengono due misure relative alla lunghezza dei layer.
Spessore medio	Due misure relative allo spessore medio dei due layer
Spessore minimo	Due misure relative allo spessore minimo dei due layer
Spessore massimo	Due misure relative allo spessore massimo dei due layer
Linear fit error	Per ogni layer, lo spessore è calcolato per ogni riga che compone l'immagine, e se ne considera per ognuna il valor medio. Per ottenere informazioni sull'allineamento delle cellule che compongono il layer, si approssimano i veri punti medi a una linea, e se ne valuta l'errore quadratico medio.

Fig. 7 Tabella riassuntiva dei valori ottenuti con l'analisi topologica

I valori ottenuti alla fine dell'analisi sono raccolti in una matrice le cui dimensioni sono:

*1*12*Ultimo giorno in cui la slice è stata fotografata.mat*

Il datamatrix è automaticamente salvato nella cartella "DatamatrixSlice", che viene creata, nel momento in cui si avvia l'analisi topologica, nel folder il cui è salvato il .exe di Ne.Mo..
Nell'ottica di rendere più user-friendly la GUI, si è aggiunta una finestra che, prima dell'inizio dell'analisi, informa l'utente di selezionare due punti, che individuano una retta tale da separare i due layer presenti nell'immagine binaria. Inoltre, si è aggiunto un messaggio che informa l'utente che l'analisi è terminata, e che il datamatrix è salvato nell'apposita cartella.

3.1.5 Creazione del datamatrix globale

La prima versione del software, presentata nel 2007, e quella immediatamente successiva, nel 2009, presentavano un grande svantaggio: non era stato implementato un editor che permettesse all'utente di selezionare i datamatrix per le singole cellule e automaticamente creasse il datamatrix globale, cioè la matrice tridimensionale. E' necessario, per realizzare un confronto tra tutti i campioni e trovare delle possibili analogie, racchiudere all'interno di un datamatrix globale i singoli datamatrix, concatenandoli tra loro. Tali matrici sono bidimensionali, come si è già sottolineato nei precedenti paragrafi, cioè hanno tante colonne quante sono le variabili misurate e tante righe quanti sono gli istanti di tempo in cui le variabili sono state estratte, cioè i giorni in cui le cellule sono state osservate e fotografate al microscopio.
I datamatrix sono come delle "slice" che vengono sovrapposte ottenendo un datamatrix tridimensionale, che avrà dimensioni

*Numero di variabili estratte*numero di cellule che si vogliono analizzare*numero di giorni*

come mostrato in Fig. 8

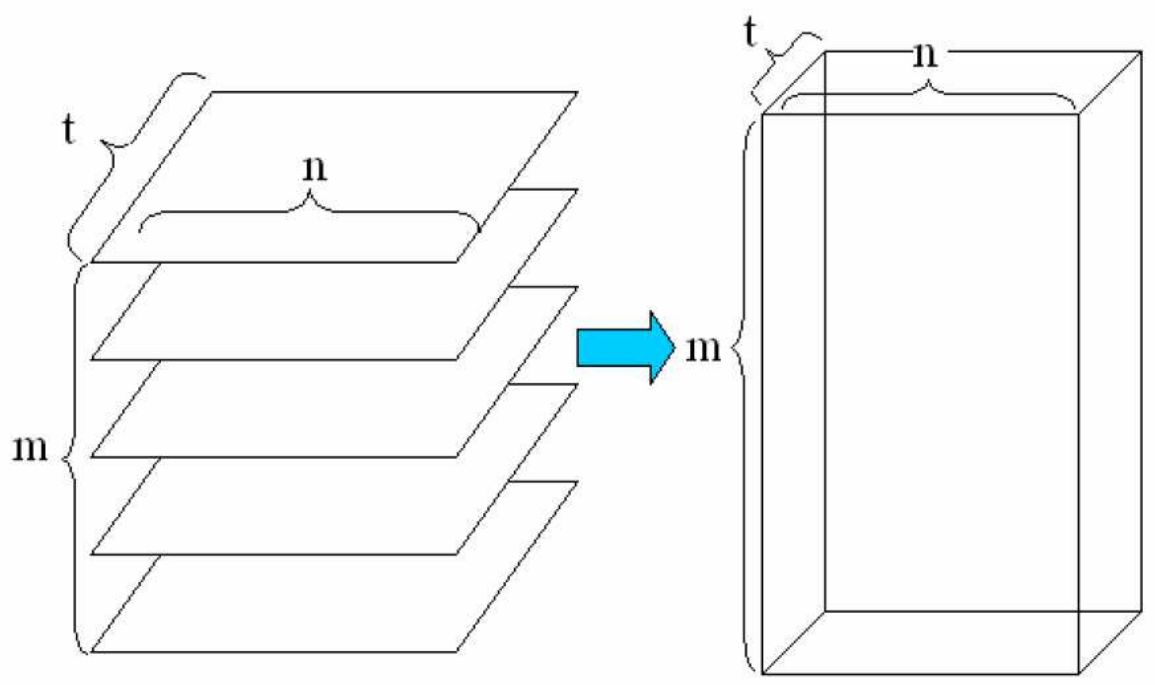


Fig. 8 Costruzione del datamatrix globale

La difficoltà di implementazione dell'algoritmo era dettata dalla necessità di rendere la costruzione della matrice a più gradi di libertà possibile.

L'editor implementato, infatti, permette all'utente di scegliere il numero n di colture che vuole analizzare attraverso una specifica finestra che appare automaticamente una volta avviato l'algoritmo di creazione del datamatrix. A questo punto, si aprirà, per n volte, la finestra standard di ricerca dei file, e, attraverso multiselezione, l'utente può scegliere per ogni coltura il numero di datamatrix che vuole concatenare a formare il datamatrix globale. Per ogni coltura, l'algoritmo dispone i datamatrix singoli in ordine crescente del numero di cellule. Grazie alla multiselezione, non c'è nessun vincolo nella scelta di quante e quali cellule inserire nella matrice. La creazione della cartella "Datamatrix" nella directory in cui è salvato il file eseguibile facilita all'utente la selezione, perché tutte le matrici

bidimensionali hanno la stessa locazione, ma sono comunque organizzate per coltura.

Un altro problema da affrontare riguardava il tempo in cui le cellule sono fotografate: non è detto, infatti, che le cellule siano seguite per lo stesso numero di giorni. L'editor, quindi, individua, durante la multiselezione, il datamatrix singolo con la dimensione-tempo maggiore, e provvede ad aggiungere matrici di zeri ai datamatrix delle cellule seguite per tempi minori: in questo modo, la dimensione di tutte le matrici sarà la stessa, e si potrà procedere con la concatenazione delle stesse con i comandi Matlab a disposizione.

In questo modo, a differenza delle prime versioni di Ne.Mo., la creazione del datamatrix è completamente svincolata dal numero di colture, dal numero di cellule e dal numero di giorni di messa in coltura specifici.

Stesso procedimento è utilizzato anche per la creazione del datamatrix tridimensionale a partire dalle singole matrici bidimensionali ottenute dall'analisi topologica.

Il datamatrix globale verrà automaticamente salvato nella directory in cui è locato il .exe di Ne.Mo., e un messaggio a video informerà l'utente che l'operazione è avvenuta con successo.

3.1.6 Plot dei dati morfologici e topologici

A questo punto, ottenuto il datamatrix globale, l'utente può avviare la procedura di plotting dei dati. Anche in questo caso, si facilita la fruibilità degli algoritmi all'utente: infatti, appare in un primo momento una finestra che ricorda di creare il datamatrix globale, e nel caso in cui questi non l'abbia fatto, automaticamente si indirizza il flusso di programma all'editor adibito a tale compito. Successivamente, appare, nel caso dell'analisi morfologica, una GUI chiede all'utente quale tipo di analisi ha intenzione di affrontare. Infatti, come discusso in modo più ampio e articolato nell'elaborato della tesi specialistica dell'Ing. Lucia Billeci [79], è possibile scegliere tra un'analisi day-by-day, e un'analisi in dinamico.

L'informazione è salvata in un vettore, e viene utilizzata dall'algoritmo per la creazione dei grafici.

Effettuata la scelta del tipo di analisi da condurre, appare automaticamente un *pop-up menu*, in cui l'utente più scegliere le variabili da plottare.

Quando necessario, si avviano finestre per interagire nel modo più semplice possibile con l'utente. Ad esempio, si chiede per quale cellula si desidera ottenere il grafico di una particolare variabile, o se, si è interessati, ottenuto il plot per una cellula, ad ottenere lo stesso ma per un'altra cellula.

I grafici, dopo la visualizzazione a display, sono salvati automaticamente in formato bitmap in una cartella apposita, chiamata "PLOT", locata nella stessa directory in cui è presente il file eseguibile di Ne.Mo. I grafici sono organizzati in sotto-cartelle in modo ordinato; in particolare "P_Morph", per i plot ottenuti dall'analisi morfologica, con ulteriori sotto-cartelle per l'analisi day-by-day o nel tempo, e "P_Slice" per i plot ottenuti dall'analisi topologica.

Tutto ciò avviene minimizzando l'intervento dell'utente.

3.1.7 Creazione del datamatrix per la 3way-PCA

E' possibile proseguire con un'analisi di tipo multivariato, che permetta cioè di conoscere le correlazioni tra le variabili estratte nella precedente analisi morfologica. Gli algoritmi utilizzati sono stati implementati da Leardi e collaboratori in ambiente Matlab.

Prima di ciò, è necessario costruire una matrice di dati che abbia la struttura tipica del datamatrix analizzato tramite questa tecnica.

Nel caso dell'analisi morfologica, data l'elevata quantità di variabili misurate, per rendere meno complessa e più efficace l'interpretazione della PCA, sono state selezionate solamente le variabili più significative:

1. Raggio critico (Rc)
2. Numero massimo di intersezioni (Nm)
3. Coefficiente di regressione (k)

4. Percorso misurato sull'asse principale (Pa)
5. Estensione (E)
6. Area del soma (As)
7. Complemento rispetto a 360° dell'angolo del cono in cui è racchiusa la cellula (Ac)
8. Dimensione frattale (Df)

La matrice globale tridimensionale che si è ottenuta con l'algoritmo descritto dei paragrafi precedenti deve essere sostituita da t matrici di dimensioni

*Numero di variabili*Numero di cellule*

come in Fig. 9.

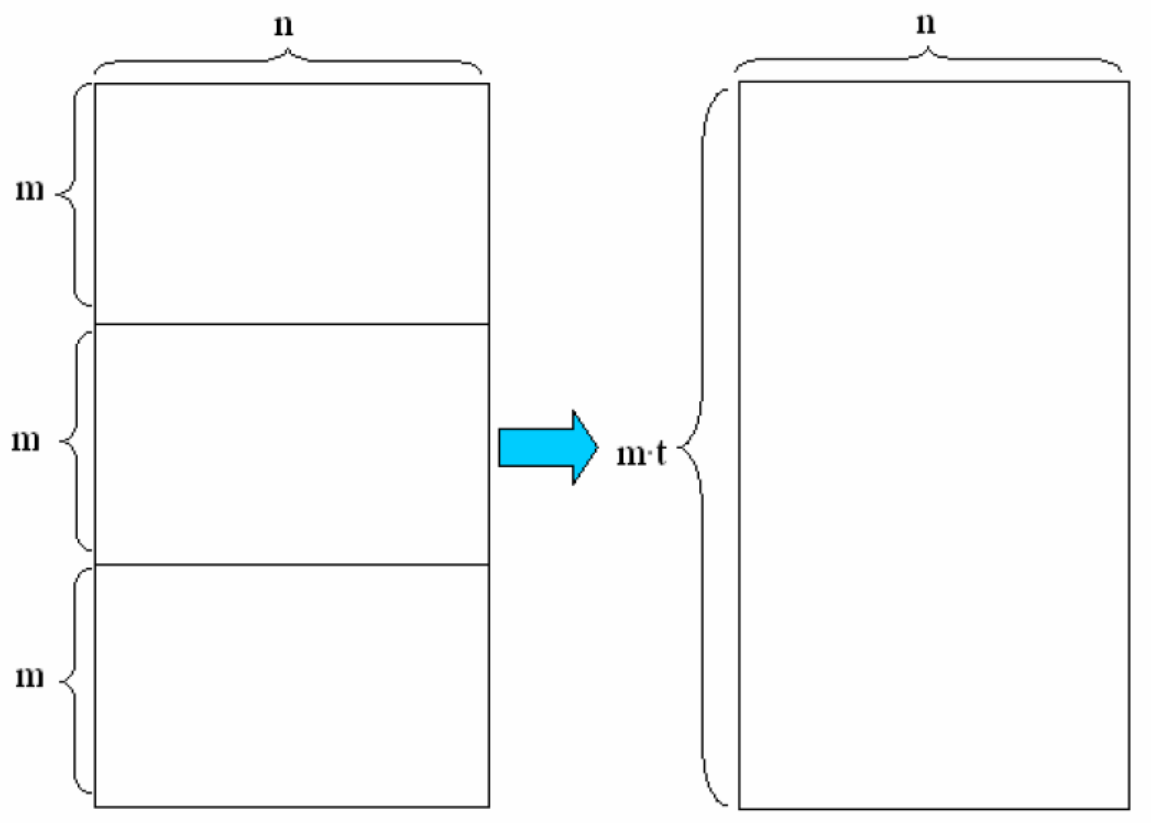


Fig. 9 Costruzione del datamatrix per la 3way-PCA

La matrice viene automaticamente salvata nella stessa directory in cui è salvata la matrice tridimensionale, e un messaggio a video informa l'utente dell'avvenuta creazione e salvataggio della stessa.

3.1.8 3-way PCA

Si discute adesso la seconda tecnica utilizzata per l'analisi dei dati estratti dall'analisi morfologica delle cellule neuronale, cioè la 3-way PCA: questa è particolarmente importante per conoscere le correlazioni tra le variabili estratte. La tecnica sarà descritta in maniera più dettagliata perché è stata implementata nella seconda fase della tesi, parte più consistente del lavoro, in linguaggio di programmazione Java.

Come è stato già detto nei precedenti paragrafi, gli algoritmi sono stati implementati in Matlab da Leardi e collaboratori.

La 3-way PCA è una tecnica di analisi multivariata che deriva dalla PCA, molto nota ed utilizzata per l'analisi dei dati che provengono dall'osservazione di un certo numero di variabili su un certo numero di soggetti od oggetti. Sia la PCA che la 3-way PCA hanno lo scopo di riassumere tutte le informazioni contenute all'interno di un datamatrix in poche componenti, e farlo in modo efficace. Una riduzione delle componenti da analizzare è fondamentale nel caso in cui una descrizione completa delle correlazioni richiederebbe un'analisi su un elevatissimo numero di dati. La differenza sostanziale tra le due tecniche è che mentre la PCA viene applicata su una matrice di dati bidimensionale, la 3-way PCA è stata modellata per lo studio di dati organizzati in una matrice tridimensionale. Spesso, infatti, le variabili sono misurate in diverse condizioni, ad esempio, in diversi istanti temporali, per cui è necessario aggiungere una terza dimensione alla matrice.

Una matrice tridimensionale può essere analizzata analizzando con la PCA le varie matrici bidimensionali che la compongono, ma la forza della 3-way PCA è la capacità di analizzare dati nelle loro correlazioni sulle tre dimensioni, quindi di giungere a conclusioni più complete ed esaustive.

Sono stati proposti diversi modelli di analisi 3-way PCA, ma in questo lavoro di tesi e nei precedenti è stato utilizzato il modello Tucker3, che risulta essere quello più generale.

Il diagramma di flusso è riportato in Fig. 10 [81]. Per informazioni più dettagliate si rimanda al punto [79] della bibliografia.

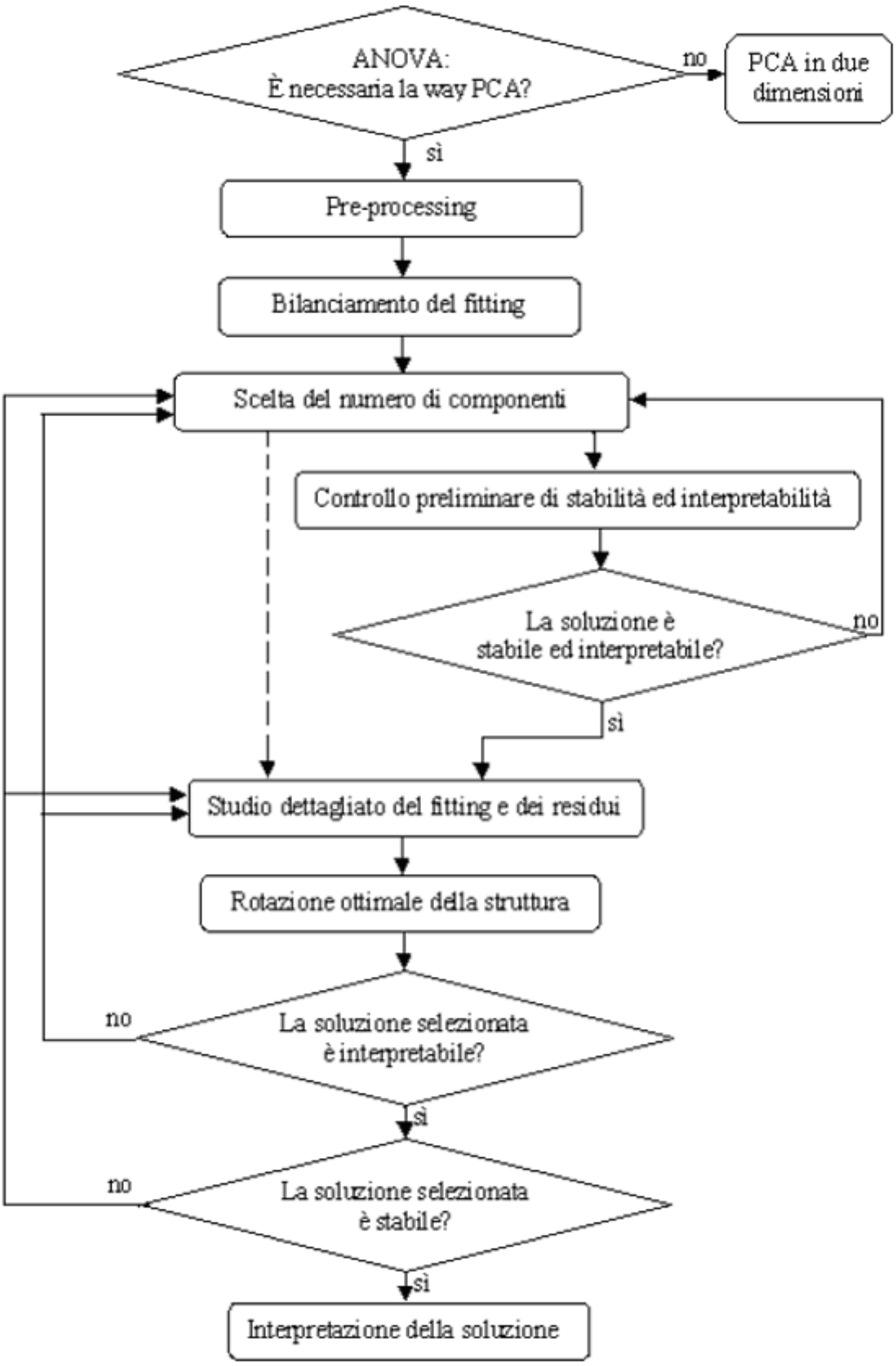


Fig. 10 Diagramma di flusso della 3-way PCA, modello Tucker 3

A questo editor, sono state aggiunte delle finestre che permettessero all'utente di scegliere se effettuare o meno lo *scaling* della matrice tridimensionale di partenza, e di inserire il numero di condizioni, di solito coincidenti con la terza dimensione della matrice, e il numero di iterazioni.

Terminato il debug e le modifiche per rendere il tool più user-friendly, si è ottenuto un file *standalone*, un file cioè che può essere eseguito su un qualsiasi computer senza la necessità che ci sia istallato Matlab.
Per generare un file eseguibile a partire da una sorgente Matlab, è sufficiente usare il compilatore c/c++ di Matlab, che trasforma il .m in .c e quindi lo compila utilizzando un compilatore interno. Tutto ciò è relativamente semplice da implementare: si tratta, infatti, di un comando da eseguire sul workspace di Matlab, che converte programmi scritti in ambiente Matlab in modo da poter essere usati anche da utenti che non hanno la licenza MathWorks. Il file eseguibile, inoltre, è utilizzabile con diversi sistemi operativi: ciò è un vantaggio che ne aumenta la fruibilità.

La versione definitiva del software si è registrata presso la SIAE con il nome NEMO: Neuron Morphological Analysis Tool 1.0 [82]. Nonostante ciò, non si hanno diritti di utilizzazione economica: è un software, quindi, *open-source*.
In Fig. 11 si riporta la tabella, già presentata nel Capitolo 1, con l'aggiunta di una riga per la descrizione delle potenzialità di Ne.Mo.: in questo modo, si facilita la comparazione del tool agli altri software discussi precedentemente.

Software		Sholl Analysis	Fractal Dimension	Soma Area	Neurite length	Neuron count	3-way PCA	Type of cells	Principal Advantages	Principal defects
Neurolucida		Yes	No	No	Yes	Yes	No	Not only neurons; Fluorescent or not.	Good resolution;	Not free; Only Windows.
Syn-D		Yes	No	Yes	Yes	Yes	No	Fluorescent	Free; Automated; Time-efficient; Precise.	Only Windows and MacOS.
ImageJ	**NeuronJ**	No	No	No	Yes	No	No	All neurons.	Free	Only tracing, no analysis.
	NeuronTracer	No	No	No	Yes	No	No	Specific marker; Confocal images.	Free; Totally automated.	Only tracing, no analysis.
	Neurphology	No	No	Yes	Yes	No	No	All neurons.	Free; Automated; Time-efficient.	Incompleted analysis
	NeuronMetrics	No	No	Yes	Yes	No	No	All neurons.	Free; SemiAutomated.	Incompleted analysis
	Sholl Analysis	Yes	No	No	Yes	No	No	All neurons.	Free; Automated.	Only Sholl Analysis.
	FracLac	No	Yes	No	No	No	No	All neurons.	Free; Automated.	Only Fractal Dimension.
	Neuron_Morpho	No	No	No	Yes	No	No	Confocal images	Free;	Not Automated.
	Cell counter	No	No	No	No	Yes	No	All images.	Free.	Only count.
Neuron Studio		Yes	No	No	Yes	No	No	Confocal images	Free; 3D reconstruction.	Only confocal Images.
MetaMorph		Yes	No	Yes	Yes	No	No	All neurons	Accurated.	Not free;
TREE toolbox		No	No	No	Yes	No	No	All neurons.	Good neurite reconstruction.	Need Matlab; Only tracing.
Ne.Mo.		Yes	Yes	Yes	Yes	Yes	Yes	All neurons	Free; Accurated; Multivariate analysis	Preliminary folders organization.

Fig. 11 Analogie e differenze tra Ne.Mo. e altri software reperibili in letteratura

3.2 Plug-in Java per la 3way-PCA

Come si è già descritto nel Capitolo 2, Image J è un programma informatico di elaborazione digitale delle immagini, rilasciato nel pubblico dominio, sviluppato dal National Institute of Health degli Stati Uniti e scritto in linguaggio di programmazione Java. Image J possiede un ricco editor di plug-in, molti dei quali sono scritti dagli utenti stessi: essi rendono possibile risolvere molti problemi, soprattutto di processing delle immagini e di visualizzazione delle cellule in 3D.

Data la possibilità di poter condividere plug-in con la comunità di Image J, si è pensato di tradurre in linguaggio di programmazione Java gli algoritmi che implementano la 3way-PCA scritti da **Leardi** e collaboratori in Matlab.

Dopo una breve panoramica sul linguaggio Java, si passerà a descrivere gli algoritmi che implementano la 3way-PCA.

3.2.1 Il linguaggio di programmazione Java [82][83]

Java è un linguaggio di programmazione orientato agli oggetti creato dagli ingegneri della Sound Microsystems. La programmazione orientata agli oggetti è un particolare paradigma di programmazione che permette di definire degli oggetti-software in grado di interagire gli uni con gli altri attraverso lo scambio di messaggi. L'oggetto non è altro che una regione di memoria allocata, che agisce come un fornitore di messaggi, detti metodi, che il codice eseguibile del programma attiva su richiesta.

Nel linguaggio Java, gli oggetti sono dotati di metodi. Questi sono abitualmente usati per implementare agevolmente molti altri costrutti che alcuni altri linguaggi forniscono nativamente. Inviare un messaggio ad un oggetto significa invocare un metodo su quell'oggetto. Il metodo riceve come parametro l'oggetto con cui è stato invocato, e sull'oggetto esegue determinati calcoli.

Inoltre, in Java non esistono funzioni: i blocchi di codice che non appartengono a nessun oggetto sono implementati come metodi detti statici.

Le differenze con Matlab sono sostanziali.

Matlab è ambiente di calcolo numerico che comprende anche l'omonimo linguaggio di programmazione ad alto livello, creato dalla MathWorks, e che fondamentalmente manipola matrici. All'interno dell'ambiente Matlab, sono disponibili numerosi tool a supporto dei più disparati campi di studio. In particolare, sono implementate molteplici funzioni con le matrici: dall'inversa, alla moltiplicazione, a operazioni più complesse come le decomposizioni.

In Java, al contrario, le uniche operazioni presenti sono quelle algebriche semplici: è spesso necessario, quindi, creare dei metodi, cioè dei sottoprogrammi da richiamare se necessario per ottenere specifici task.

In genere, in un programma Java vengono utilizzate molte classi, che possono essere:

- Classi fornite con l'ambiente di sviluppo;
- Classi precedentemente sviluppate dal programmatori o da altri, disponibili spesso sul web;
- Classi che compongono il programma.

Queste classi sono organizzate in librerie che, nel caso di Java, prendono il nome di *package*. Se in un programma Java si vuole utilizzare una classe che appartiene a un package, bisogna informare esplicitamente compilatore e la macchina virtuale, attraverso una specifica direttiva di importazione [84][85].

A questo punto, si discuterà di un particolare tipo di oggetto utilizzato nell'implementazione della 3-way PCA: l'array.

Un oggetto array (o semplicemente array) contiene una collezione di elementi dello stesso tipo, ognuno dei quali è indicizzato, ovvero identificato, da un numero. Una variabile di tipo array contiene riferimenti ad oggetti array.

Per poter utilizzare un array in Java, è necessario dichiarare una variabile di tipo array che permetta di riferirsi ad un oggetto array, costruire quest'ultimo specificandone le dimensioni, quindi il numero di elementi, e infine accedere mediante la variabile array agli elementi dell'oggetto array per assegnarli o leggerli come fossero

singole variabili. Per fare ciò, ci sono specifici comandi da utilizzare, ognuno con una specifica sintassi da rispettare.

Per quanto riguarda le matrici, esse sono una collezione in forma tabellare di elementi dello stesso tipo, ognuno dei quali è indicizzato da una coppia di numeri che identificano riga e colonna dell'elemento. Una matrice può essere implementata in Java mediante un array i cui elementi sono a loro volta riferimenti ad array che rappresentano le singole righe di una matrice. Un esempio di ciò è schematizzato nella Fig. 12 [86].

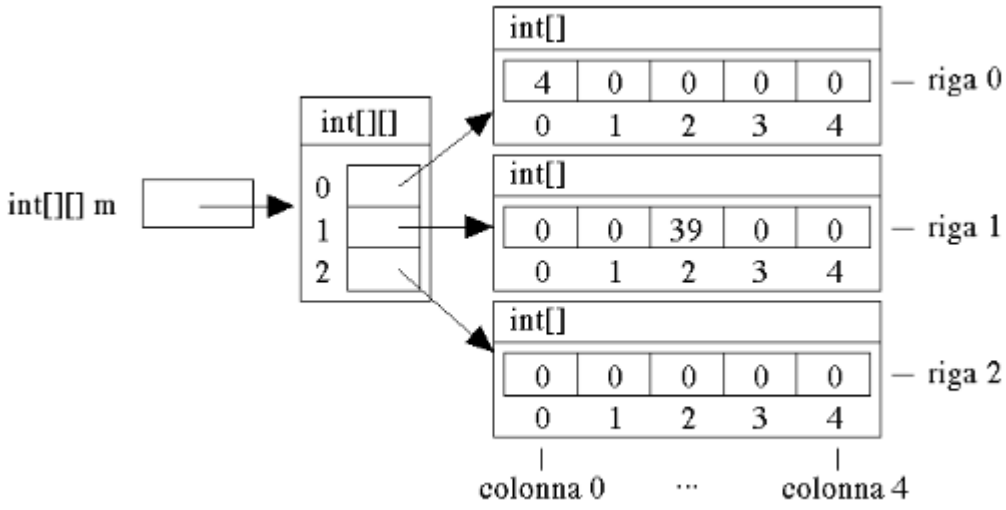


Fig. 12 Esempio di costruzione di una matrice in Java

A questo punto, è possibile sottolineare un'altra differenza con Matlab: mentre in Java si implementano array di array per ottenere matrici, in Matlab si implementano matrici vere e proprie. Inoltre, l'indicizzazione è diversa: per array di dimensione n, in Matlab gli indici vanno da 1 a n, in Java, come in molti altri linguaggi di programmazione, ad esempio il C, vanno da 0 a n-1. Inoltre, mentre in Matlab la dimensione della matrice non deve essere indicata a priori, ma è possibile aggiungere righe e colonne in maniera piuttosto semplice con operazioni di incolonnamento, o con opportuni comandi (come vertcat), in Java al momento della dichiarazione di una variabile di tipo array, è opportuno creare l'oggetto array stesso, indicandone le dimensioni. Nel caso in cui questo non sia possibile, per ragioni di implementazione dello specifico algoritmo, è possibile

dichiarare una lista di elementi, in modo da non aver il vincolo della lunghezza massima prefissata.

3.2.2 Strategie di implementazione del plug-in

Gli algoritmi che eseguono la 3-way PCA utilizzati in Ne.Mo. sono stati implementati, come più volte sottolineato, da Leardi e collaboratori: era quindi disponibile il codice sorgente della tecnica di analisi multivariata, scritto in ambiente Matlab. Questo si compone di un *main*, che implementa il metodo Tucker3, che richiama due funzioni, implementate in altrettanti editor.

Per quanto riguarda Java, non esistono le funzioni, quindi è stato necessario implementare dei metodi. Oltre ai metodi che eseguono gli algoritmi implementati nelle due funzioni Matlab, ne sono stati scritti altri, poiché in Java molti comandi esistenti in Matlab non sono presenti. Era necessario, infatti, creare dei comandi che permettessero all'utente di inserire dei parametri necessari all'elaborazione, ma anche di creare dei sottoprogrammi che implementassero, ad esempio, il prodotto tra matrici, operazione molto usata nella stesura della 3-way PCA: è utile quindi implementarla solo una volta in un metodo, che poi si richiama all'occorrenza.

Sono stati, quindi, implementati, i seguenti metodi:

- *getImportData*, che permette all'utente la selezione della matrice su cui si vuole effettuare l'analisi;
- *showDialog*, che crea una *dialog box* per l'inserimento di alcuni parametri indispensabili per la tecnica di analisi implementata;
- *dialogItemChanged*, che effettivamente acquisisce i parametri inseriti dall'utente nella *dialog box* creata con il metodo precedente;
- *round*, che permette di approssimare un numero double a una certa cifra decimale, indicata dal programmatore. Matlab arrotonda di default alla terza cifra decimale, Java alla sedicesima: si è dovuto, quindi, implementare questo tipo di

metodo per evitare che le condizioni nei vari cicli while presenti nell'algoritmo (necessari per descrivere la caratteristica iterativa della tecnica) fossero diverse, e quindi il ciclo si interrompesse in tempi diversi;

- *getMean*, che, dato un array, ne calcola ma media tra gli elementi;
- *getMax* e *getMin*, che, dato un array, ne individua l'elemento massimo e minimo rispettivamente;
- *getMatrixProduct*, che implementa il prodotto tra due matrici;
- *getMatrixTraspose*, che ritorna la trasposta della matrice data in ingresso;
- *kron*, che ritorna il tensore prodotto di kronecker tra le matrici x e y, contenente al suo interno tutti i possibili prodotti tra gli elementi di x e y stesse;
- *sqrtMatrix*, che, data una matrice x, ne ritorna una y, delle stesse dimensioni di x, tale che y*y=x. Questo metodo è stato particolarmente difficile da implementare: c'era bisogno, infatti, di ottenere la decomposizione di Schur dalla matrice di partenza. In algebra lineare, tale decomposizione è un importante processi di fattorizzazione di una matrice: data una matrice quadrata A di numeri in generale complessi, questa può essere decomposta come $A=QUQ^*$, in cui Q è una matrice unitaria, Q^* denota la trasposta coniugata di Q e U denota la matrice triangolare superiore le cui entrate diagonali sono esattamente gli autovalori di A. Implementare tutti questi metodi che facessero da supporto al metodo sqrtMatrix risultava essere complesso: si è quindi cercata in rete una libreria che permettesse questo tipo di decomposizione.

Gli algoritmi implementati in questo lavoro di tesi sono riportati interamente nell'Appendice A.

In prima battuta, è necessario permettere all'utente di selezionare la matrice da analizzare: per fare ciò, si è importata una specifica libreria, che permette di leggere file .mat (formato standard in cui in Matlab vengono salvate le matrici). Subito dopo l'avvio del plug-in,

si apre la finestra standard di ricerca dei file, attraverso la quale l'utente può selezionare il datamatrix da analizzare.

A questo punto, è stato possibile implementare l'algoritmo Tucker3 vero e proprio.

Questo tipo di tecnica può essere suddivisa in vari step. Il primo step è la possibilità di eseguire il *pre-processing* dei dati, che consiste in uno *scaling* degli stessi: infatti, dato che spesso le variabili hanno scale e unità di misura diverse, sono presenti degli *offset* che possono essere eliminati centrando i dati rispetto alle variabili. Questo è implementato in Matlab con una funzione specifica, "autosc": data una matrice x, lo *scaling* ritorna una matrice ax delle stesse dimensioni di x, ottenuta dividendo la differenza tra gli elementi di una determinata colonna di x e la media su tale colonna per la deviazione standard sempre su tale colonna. In Java, non essendo disponibile un comando che facesse ciò, è stato necessario implementare il tutto. Quindi, da un semplice comando Matlab si è passati a diverse righe di codice, riportati qui di seguito:

```
double [] average = new double [nCols];
   double [] stdev = new double [nCols];

   for (int i = 0; i<nCols; i++) {
         for(int j = 0; j<nRows; j++){
               average[i] += x[j][i];
         }
         average[i] = average[i]/nRows;
   }

   for (int i = 0; i<nCols; i++){
         for (int j = 0; j<nRows; j++){
                                    stdev[i] += (x[j][i]-
   average[i])*(x[j][i]- average[i]);
         }
         stdev[i] = Math.sqrt(stdev[i]/(nRows-1));
   }

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

               x[i][j] = (x[i][j] - average[j]) / stdev[j];
         }
   }
         }
```

Successivamente, si apre un'interfaccia grafica, che permette all'utente di indicare la volontà o meno di operare lo *scaling*, il numero delle condizioni, necessario per la riorganizzazione successiva della matrice di partenza, e il numero di iterazioni, essendo la 3-way PCA una tecnica iterativa. Per questi ultimi due, sono indicati nei rispettivi *edit* i valori di default (Fig.13).

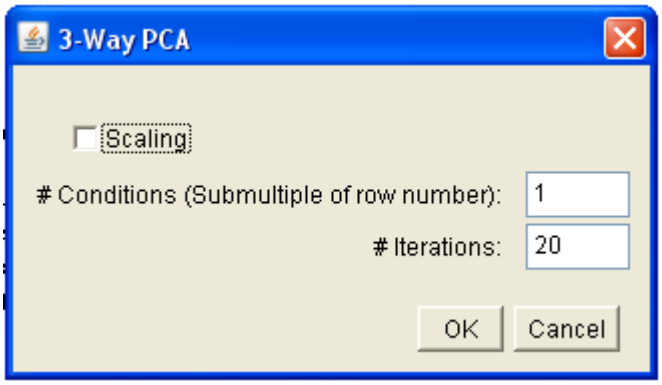


Fig. 13 Interfaccia grafica e relativi valori di default

Ottenuto il numero di condizioni, che può essere ad esempio il numero di giorni in cui una determinata coltura è osservata e fotografata, si organizzano gli elementi contenuti nel datamatrix in tre modi diversi, tali da evidenziare rispettivamente oggetti, variabili e condizioni. Si otterranno, quindi, con la riorganizzazione del datamatrix iniziale, tre matrici:

- xo, matrice degli oggetti, di dimensioni numero_oggetti * (numero_variabili * numero_condizioni);

	variabili condizione1	variabili condizione2	.....	variabili condizione c
oggetto1				
oggetto2				
.....				
oggetto n				

- xv, matrice delle variabili, di dimensioni numero_variabili * (numero_oggetti * numero_condizioni);

	oggetti condizione1	oggetti condizione2	.....	oggetti condizione c
variabile1				
variabile2				
.....				
variabile m				

- xc, matrice delle condizioni, di dimensioni numero_condizioni * (numero_oggett * numero_variabili);

	variabili oggetto1	variabili oggetto2	.....	variabili oggetto n
condizione1				
condizione2				
.....				
condizione c				

Queste in Matlab sono implementate in maniera piuttosto semplice: infatti, come sottolineato nei precedenti paragrafi, è possibile concatenare più matrici tra loro in maniera molto diretta. In Java, ciò è stato implementato elemento per elemento attraverso la concatenazione di più cicli for. Si riporta un esempio a titolo esplicativo:

Matlab:

```
for p = 1:c
  xo = [xo x((p-1)*o+1:p*o,:)];
end;
```

Java:

```
for(int p=0; p<nConditions; p++) {
for(int i=0; i<nCols; i++) {
      for (int j=0; j<o; j++) {
            xo[j][i+p*nCols] = x[j+p*o][i];
                                    }
}
            }
```

A questo punto, si calcola la varianza iniziale dei dati: poiché la matrice iniziale è stata già pre-elaborata effettuando lo *scaling*, questa è calcolata semplicemente sommando tra loro tutti gli elementi che compongono la matrice stessa. Il risultato è disponibile all'utente: infatti, automaticamente si interrompe il flusso di programma, e appare una finestra in cui si riporta il valore ottenuto.

Dopo di ciò, si richiama un metodo, *getAutovector*, implementato da Leardi in una funzione, denominata niptuck e richiamata nel main. Questo implementa uno dei tanti metodi esistenti per calcolare gli autovettori, il NIPALS (Non-linear Iterative Partial Least Squares). E' stato inizialmente pensato per la PCA, ma può essere utilizzato anche in questo caso. E', infatti, uno dei metodi più utilizzati per il calcolo delle componenti principali da uno specifico data set. Il metodo è descritto step-by-step in Fig. 14. Questo metodo ritorna dei plot: è stato necessario, quindi, trovare un plug-in o una libreria che permettesse la visualizzazione a display dei plot. Essendo ImageJ un tool per l'elaborazione delle immagini, era già stato implementato un plug-in per il plot di dati: è stato, quindi, utilizzato per i due tipi di plot che dovevano essere visualizzati a display.

Step	Math	Explanation
1.	$\mathbf{u} := \mathbf{x}_i$	Select a column vector $\mathbf{x}_i$ of the matrix $\mathbf{X}$ and copy it to the vector $\mathbf{u}$
2.	$\mathbf{v} := (\mathbf{X}'\mathbf{u})/(\mathbf{u}'\mathbf{u})$	Project the matrix $\mathbf{X}$ onto $\mathbf{u}$ in order to find the corresponding loading $\mathbf{v}$
3.	$\mathbf{v} := \mathbf{v}/\lvert\mathbf{v}\rvert$	Normalize the loading vector $\mathbf{v}$ to length 1
4.	$\mathbf{u}_{old} := \mathbf{u}$ $\mathbf{u} := (\mathbf{Xp})/(\mathbf{v}'\mathbf{v})$	Store the score vector $\mathbf{u}$ into $\mathbf{u}_{old}$ and project the matrix $\mathbf{X}$ onto $\mathbf{v}$ in order to find corresponding score vector $\mathbf{u}$
5.	$\mathbf{d} := \mathbf{u}_{old}-\mathbf{u}$	In order to check for the convergence of the process calculate the difference vector $\mathbf{d}$ as the difference between the previous scores and the current scores. If the difference $\lvert\mathbf{d}\rvert$ is larger than a pre-defined threshold (e.g. 10^{-8}) then return to step 2.
6.	$\mathbf{E} := \mathbf{X} - \mathbf{tp}'$	Remove the estimated PCA component (the product of the scores and the loadings) from $\mathbf{X}$
7.	$\mathbf{X} := \mathbf{E}$	In order to estimate the other PCA components repeat this procedure from step 1 using the matrix $\mathbf{E}$ as the new $\mathbf{X}$

Fig. 14 passaggi fondamentali del metodo NIPALS

In primo luogo, si è implementato lo *score plot*, grafico in cui si visualizzano le coordinate per ciascun punto relative alle prime due

componenti principali, che sono quelle che di solito rappresentano in maniera efficace il fenomeno analizzato: la nuova variabile con la maggiore varianza viene proiettata sul primo asse, la variabile nuova, seconda per dimensione della varianza, sul secondo asse. Infine, si è implementato il *loading plot*, nel quale sono le variabili ad essere riportate nel nuovo sistema avente per assi le componenti principali: con questo tipo di grafico, è possibile osservare se le variabili sono simili, e pertanto forniscono lo stesso tipo di informazione, oppure sono distanti, quindi non sono simili.

Si sono eseguiti due scatter plot per ciascuna matrice, per un totale di sei grafici.

Il plug-in che permette la visualizzazione dei grafici ha varie opzioni, dalla scelta di eseguire un plot a linea continua o uno scatter plot, alla scelta delle dimensioni, del colore e della forma dello scatter plot stesso. Inoltre, c'è la possibilità di nominare gli assi, di settare i valori estremi degli assi stessi, di dare un titolo al grafico e di labelizzare, per renderne più agevole il riconoscimento, i punti plottati con un numero via via crescente: quest'ultima operazione è necessaria nel caso di datamatrix di dimensioni notevoli. Questa operazione non è banale: infatti, nei plot ottenuti con il plug-in utilizzato in questo lavoro di tesi, il centro degli assi di riferimento è posizionato nell'angolo in alto a sinistra del grafico, con l'asse x direzionato a destra, e l'asse y verso il basso: si è dovuta, quindi, effettuare una trasformazione lineare che per ogni coordinata ritornasse il valore corrispondente in questo particolare sistema di riferimento.

Oltre ai plot, il metodo ritorna le nuove variabili nel nuovo sistema cartesiano, quello delle componenti principali. Ciò è implementato per le matrici degli oggetti, delle variabili e delle condizioni, quindi per ciascuno dei tre "modi" della matrice originale.

Ottenute le coordinate nello spazio bidimensionale delle componenti principali, si è proceduto con l'implementazione della ALS (Alternative Least Square), metodo che, data una matrice x, la decompone in CS, tale che x = CS+E, in cui E è minimizzato ai

minimi quadrati. E' importante fare un attento studio dei residui: se hanno valori elevati, infatti, il modello potrebbe risultare incompleto, e bisognerà decidere se aggiungere o meno altre componenti, sempre trovando un compromesso con il cercare di mantenere una certa semplicità del modello. La soluzione del modello non è unica. Infatti, la 3-way PCA ha lo scopo di riassumere, come già sottolineato più volte, il contenuto degli oggetti, delle variabili e delle condizioni rispettivamente nelle matrici A, B e C, ma si possono trovare descrizione dei dati equivalenti ruotando queste tre matrici. Il passo successivo, quindi, consiste nell'andare a valutare per quale rotazione si ottiene la soluzione più semplice da interpretare.

Tale scelta viene effettuata eseguendo tutta una serie di rotazioni ortogonali delle tre matrici, andando a vedere per quale orientamento comune delle tre matrici il "core array" è il più diagonale possibile.

Dopo aver ottenuto una soluzione che fitta bene i dati, l'ultima cosa da fare è vedere se tale soluzione è stabile per fluttuazione dei campioni. Lo studio della stabilità è fondamentale per essere sicuri della generalizzabilità dei risultati: la soluzione scelta deve andare bene anche per campioni della stessa popolazione ma diversi da quelli su cui è stata fatta l'analisi.

La scelta della rotazione per determinare la soluzione più facile da interpretare non può essere semplice senza un'indicazione della stabilità: è per questo che è utile eseguire un controllo sulla stabilità, ruotando la soluzione. Se questa soluzione ruotata è stabile lo sarà anche quella ottenuta dopo la rotazione.

Sia la rotazione che lo studio della stabilità sono effettuati per le tre soluzioni, relative a oggetti, variabili e condizioni rispettivamente.

Una volta scelta la soluzione che meglio rappresenta i dati, è possibile interpretare i risultati dell'analisi. Si esegue quindi un ulteriore plot, con le stesse modalità del *loading plot* implementato precedentemente: si otterranno, quindi, tre grafici, per oggetti, variabili e condizioni rispettivamente. Anche in questo caso si è utilizzato, per il plot, il plug-in già disponibile in Image J. Si è ottenuto quindi uno *scatter plot*, e per rendere più semplice

l'individuazione dei tre modi diversi dell'analisi, si sono colorati i punti individuati nel plot in tre colori diversi: rosso per gli oggetti, verde per le variabili e blu per le condizioni. Questi rappresentano i grafici più significativi di tutta l'analisi.

Infine, si calcola, per i tre modi della matrice, l'RMSE (Root Mean Square Error) come deviazione standard degli scarti. C'era bisogno di un *bar plot*, che è implementato in Matlab con una specifica funzione. Non era, invece, disponibile un equivalente in Image J, nemmeno nel plug-in utilizzato per i grafici descritti precedentemente: si è quindi dovuto prima di tutto creare un vettore, della stessa dimensione n del residuo da plottare e i cui elementi sono numeri crescenti da 1 a n. Si riporta, a titolo d'esempio, il vettore "obj" creato per il modo-oggetto dell'analisi:

```
double [] obj;
obj = new double [ro.length];
for(int i = 0; i<ro.length; i++){

      obj[i] = i+1;
            }
```

Si è quindi utilizzata la sintassi canonica del plug-in già disponibile in Java per ottenere un grafico, come *scatter plot*, del residuo in funzione del vettore calcolato con le modalità descritte precedentemente. A questo punto, per creare le barre tipiche del *bar plot*, si è utilizzata un'opzione presente nel plug-in: è possibile disegnare delle linee sul grafico, dando una coppia di coordinate, che rappresentano gli estremi della linea stessa. I segmenti che si tracceranno automaticamente sul grafico collegheranno ciascun punto dello *scatter plot* alla proiezione ortogonale sull'asse x, creando la rappresentazione grafica tipica del plot a barre. Per far ciò per ogni elemento del vettore dei residui, si è implementato un ciclo for, di cui si riporta un esempio:

```
for (int i = 0; i<ro[0].length; i++){
        plot6.drawLine(i+1, ro[i][0], i+1, 0);
            };
```

Lo stesso è stato implementato anche per variabili e condizioni.

A questo punto, si è operato un debug con una matrice di prova per essere sicuri che il plug-in funzionasse correttamente. Infine, si è inserito il codice sorgente in un'apposita cartella di Image J, seguendo le istruzioni reperibili sul sito del tool stesso: in questo modo, cliccando su Plugins→ Three way PCA, è possibile effettuare l'analisi (Fig. 15)

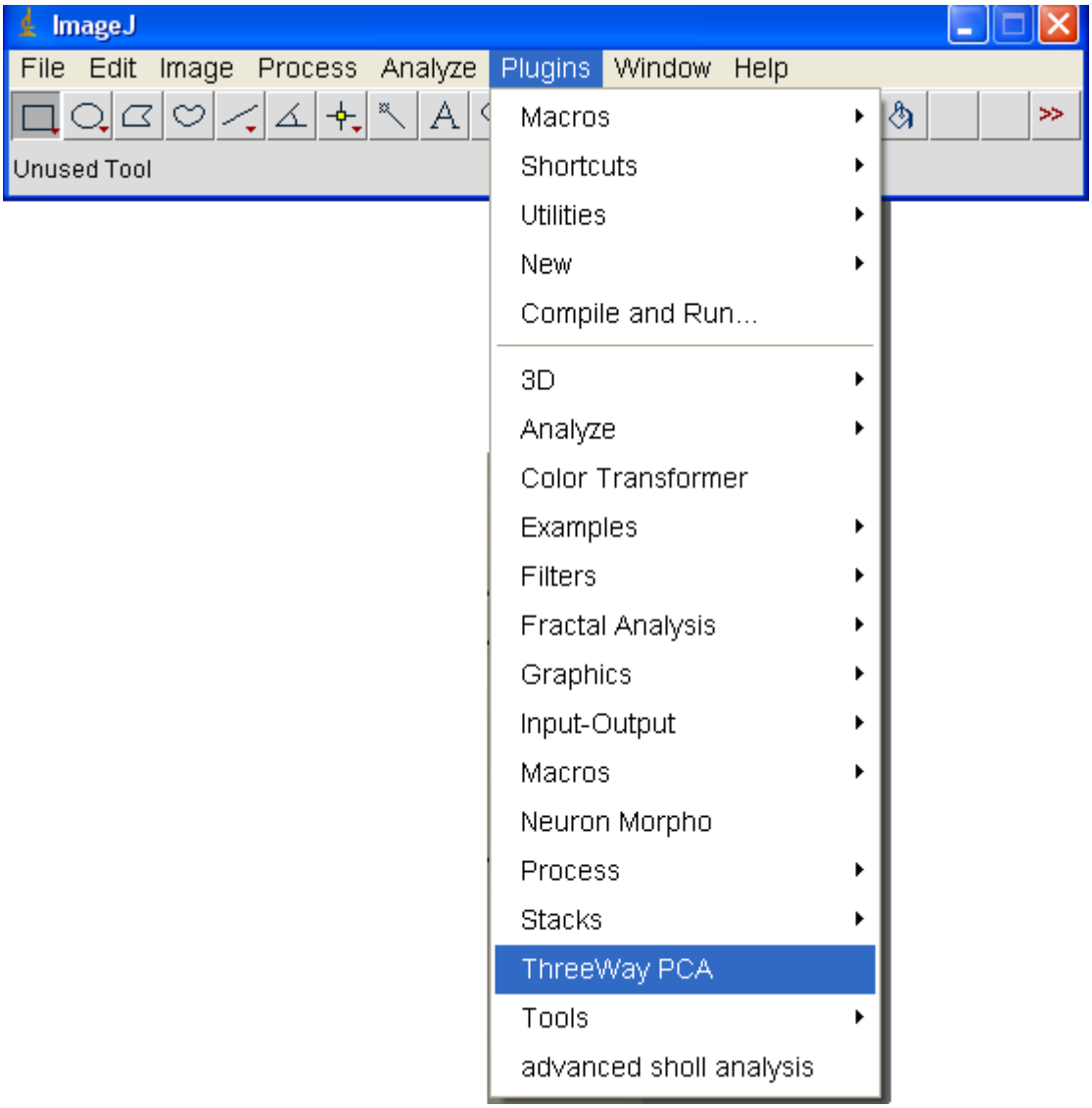


Fig. 15 Menu a tendina di Image J che permette l'avvio della 3-way PCA

Capitolo 4

Presentazione dei risultati

In questo capitolo sono riportati i risultati ottenuti utilizzando le tecniche presentate nel capitolo precedente.

Si è innanzitutto validato il plug-in che effettua la 3-way PCA, scritto in linguaggio di programmazione Java. Per far ciò, si sono analizzati diversi datamatrix sia con l'algoritmo implementato da Leardi, sia con quello implementato in questo lavoro di tesi: si riportano di seguito i risultati.

E' importante, per essere certi che il plug-in in Java e l'editor in Matlab diano esattamente gli stessi risultati, analizzare più datamatrix: nell'algoritmo, infatti, sono presenti strutture di "controllo a condizione", ossia particolari strutture sintattiche grazie alle quali una data istruzione o un dato blocco di istruzioni sono eseguiti solo se vale una certa condizione. Le condizioni possono essere diverse a seconda della matrice utilizzata, quindi per saggiare la validità di tutte le righe di codice è necessario applicare gli algoritmi a diversi datamatrix.

Per quanto riguarda i dati analizzati, due delle matrici analizzate contenevano parametri morfologici estratti da fotografie di cellule di Purkinje osservate a diversi intervalli di tempo: la prima, ottenuta durante il lavoro di tesi specialistica dell'Ing. Lucia Billeci [79], contiene esclusivamente cellule estratte da cervelli di topi wild-type, la seconda, ottenuta durante il lavoro di tesi triennale di Margherita Brancadoro [80], analizza due colture neuronali, una proveniente da

topi wild-type e l’altra da topi modelli animali di autismo, in particolare Engrailed 2.

Inoltre, verrà riportato in maniera dettagliata ed approfondita ciò che si è osservato e dedotto dall’analisi delle cellule neuronali nel claustro, sottile lamina di sostanza grigia situata nella corteccia dell'*insula*. Si sono fotografati neuroni su vetrini fissati contenenti le *slice* di claustro, e se ne è effettuata l’analisi morfologica con Ne.Mo. Organizzati i dati in una matrice di dimensioni opportune, si è effettuata l’analisi multivariata, quindi la 3-way PCA, sia con Ne.Mo. che con il plug-in per Image J.

Infine, anche per sottolineare come questo tipo di analisi multivariata si possa utilizzare nei più disparati campi di ricerca, si sono analizzati dati ottenuti da esperimenti su tre tipi cellulari diversi in coltura sottoposti a una particolare nanoparticella, Ag NM300, che si suppone tossica. Si sono effettuati diversi test per verificare la vitalità della coltura cellulare a varie concentrazioni della nanoparticella stessa, e i dati sono stati infine organizzati in un datamatrix e analizzati con la 3-way PCA.

4.1Analisi multivariata su neuroni

4.1.1 Costruzione del datamatrix

Come più volte ribadito, preliminarmente all’analisi tramite tecnica 3-way PCA è necessario organizzare i dati ottenuti tramite l’analisi morfologica.

Il datamatrix che si ottiene dall’analisi morfologica effettuata con Ne.Mo. ha dimensioni n*m*c, in cui n è il numero di cellule analizzate, m il numero di variabili estratte e c rappresenta le condizioni in cui i neuroni sono osservati e, quindi, fotografati. Pertanto, la matrice che si ottiene è tridimensionale, e contiene al suo interno dati di tutte le cellule per ciascuna condizione.

Le condizioni a cui le cellule sono osservate possono essere le più disparate: in questo caso, sono i diversi giorni in cui le cellule sono fotografate.

Il datamatrix per la 3-way PCA, invece, deve essere costituito da c matrici n*m concatenate lungo la prima dimensione: si ottiene, così, una matrice bidimensionale (n*c)*m.

Data l'elevata quantità di variabili estratte con l'analisi morfologica effettuata con il tool Ne.Mo., per rendere meno complessa e più efficace l'interpretazione della 3-way PCA, sono state selezionate solamente i parametri più significativi, che si riportano di seguito, insieme alle abbreviazioni con cui questi stessi verranno indicati successivamente nei grafici:

1. Raggio critico (Rc)
2. Numero massimo di intersezioni (Nm)
3. Coefficiente di regressione (k)
4. Percorso misurato sull'asse principale (Pa)
5. Estensione (E)
6. Area del soma (As)
7. Complemento rispetto a 360° dell'angolo del cono in cui è racchiusa la cellula (Ac)
8. Dimensione frattale (Df)

Per ogni datamatrix analizzato, si riporteranno le dimensioni e la condizione specifica a cui le cellule sono state sottoposte.

4.1.2 Analisi dei grafici

4.1.2.1 Cellule wild-type

La prima matrice di dati analizzata, ottenuta dall'Ing. Lucia Billeci durante il suo lavoro di tesi specialistica, contiene informazioni morfologiche di neuroni, provenienti da due diverse colture di topi wild-type, fotografate in diversi giorni.

Le condizioni scelte sono i giorni di osservazioni comuni alle due colture, che risultavano essere uguali a 5: dall'undicesimo al quindicesimo giorno di coltura.

Le cellule considerate, invece, sono 20, di cui 8 della prima coltura e 12 della seconda.

In definitiva, quindi, il datamatrix ottenuto aveva dimensioni (20*5)*8.

Si è analizzato il datamatrix così ottenuto con il plug-in in linguaggio di programmazione Java, e se ne sono confrontati i grafici con quelli ottenuti dall'Ing. Lucia Billeci, che analizzò lo stesso con gli algoritmi implementati in ambiente Matlab da Leardi.

Applicando gli algoritmi che implementano il modello Tucker3, si ottengono tre grafici che rappresentano rispettivamente:

1. Piano delle variabili
2. Piano degli oggetti (cellule)
3. Piano delle condizioni (giorni)

Vengono riportati di seguito i grafici, tre ottenuti nello scorso lavoro di tesi e tre con il plug-in in Java, per dimostrare che effettivamente si estraggono le medesime informazioni: questi poi verranno commentati e interpretati uno alla volta (Fig. 1-3)

Si ricorda che, mentre in Matlab la numerazione dei vettori va da 1 a n, in Java va da 0 a n-1: ciò si ripercuote anche sulla numerazione dei punti negli scatter plot.

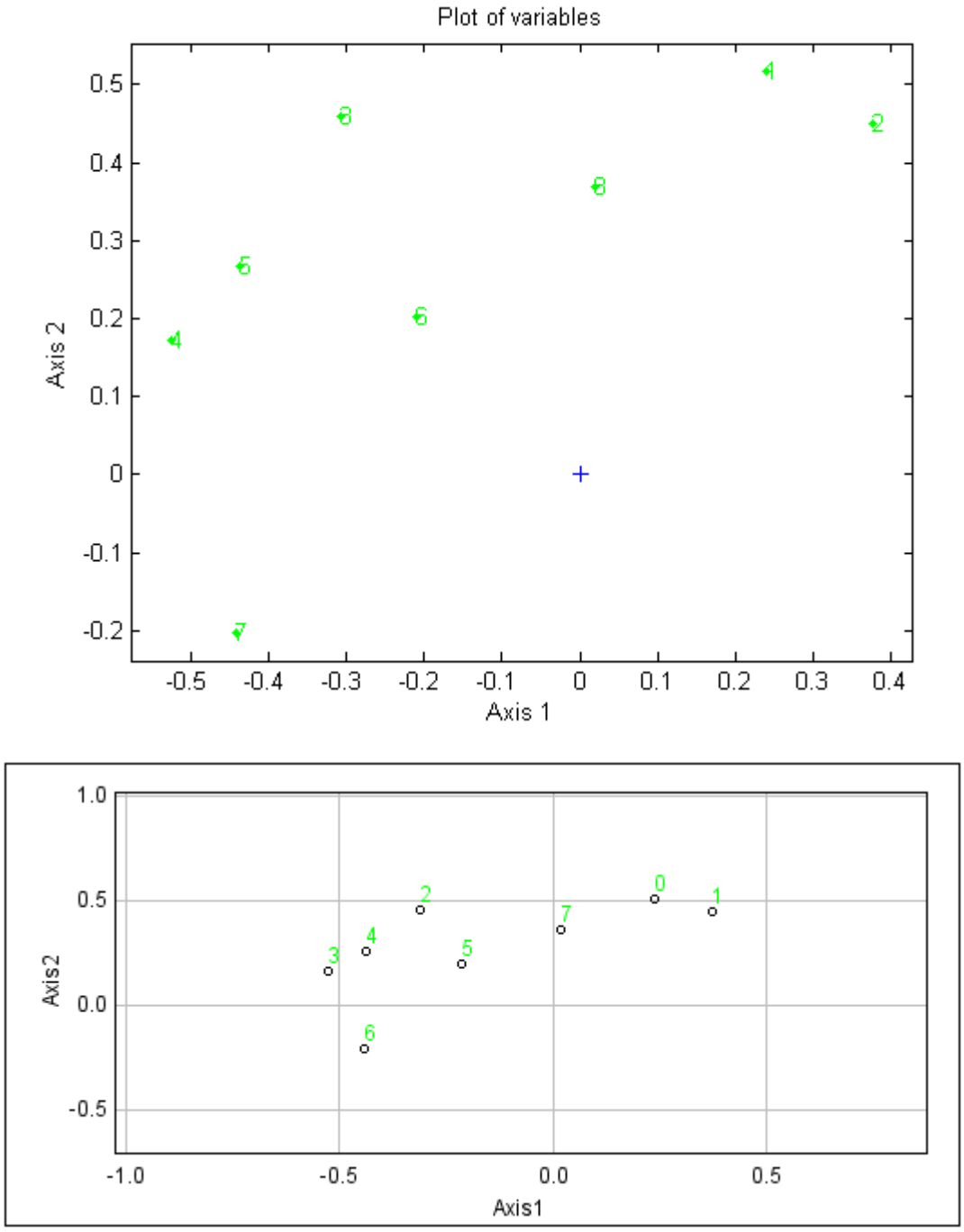


Fig. 1 Grafico delle variabili

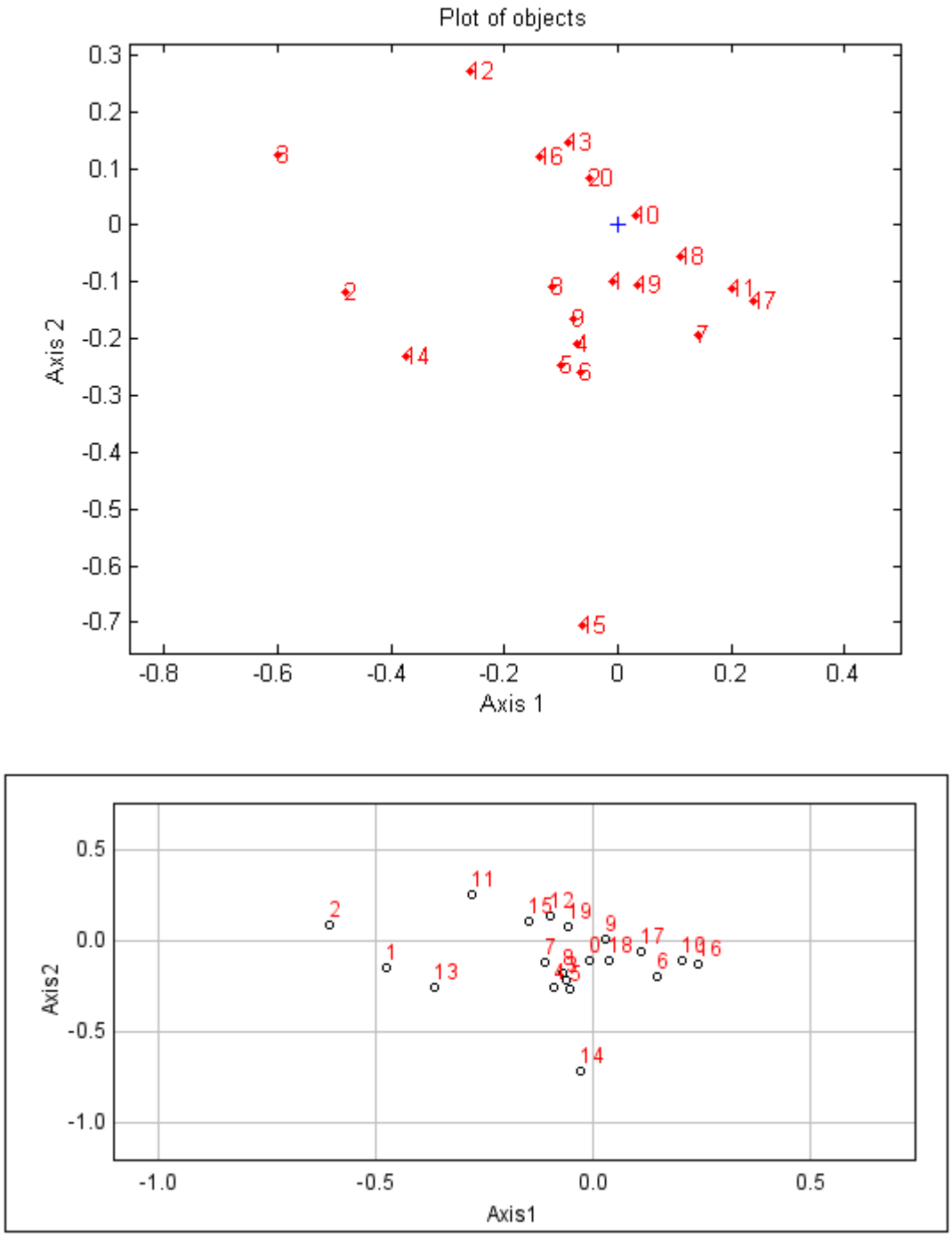


Fig. 2 Grafico delle cellule (oggetti)

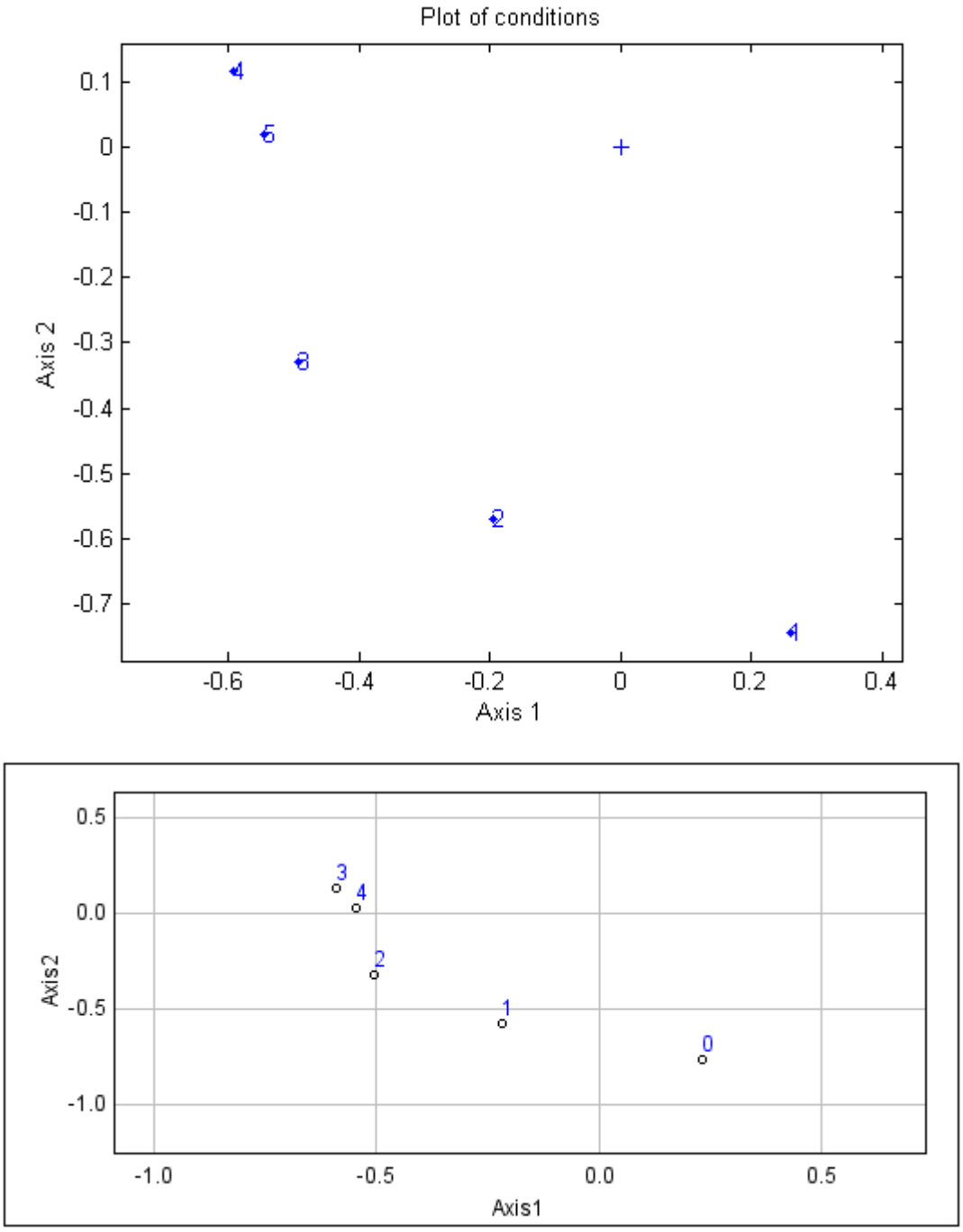


Fig. 3 Grafico dei giorni (condizioni)

I grafici danno la stessa informazione, siano essi ottenuti con l'editor Matlab o con il plug-in in Java: quest'ultimo, quindi, è stato implementato in maniera corretta.

Per dare un senso ai grafici ottenuti, è necessario dare un'interpretazione agli assi, cioè capire a che variabili essi possano corrispondere.

In questo caso, l'asse x rappresenta la direzionalità con verso crescente da destra verso sinistra, mentre l'asse y rappresenta la complessità dell'albero dendritico con verso crescente dal basso verso l'alto. Questa interpretazione viene giustificata di seguito, analizzando i tre grafici.

► Grafico delle variabili

Si riporta, in Fig. 4, il grafico delle variabili, con gli assi rinominati in base all'interpretazione data agli stessi. Per facilitare la comprensione del grafico, si sono riportati, in corrispondenza dei singoli punti, l'abbreviazione delle variabili morfologiche a cui essi corrispondono.

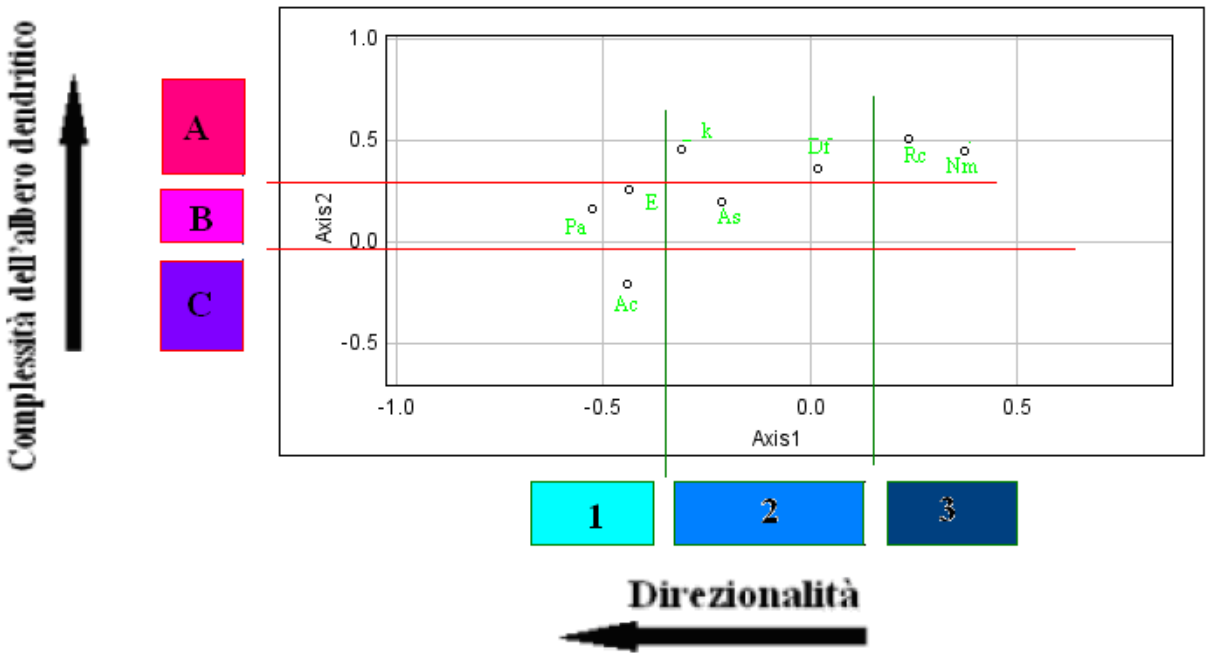


Fig. 4 Grafico delle variabili, interpretato in base al significato degli assi

Si nota che le variabili sono distribuite sia lungo l'asse x che lungo l'asse y.

In particolare, considerando la distribuzione lungo l'asse x, è possibile individuare tre *cluster*:

1. Cluster 1(Estensione, Percorso sull'asse, Angolo del cono). Queste variabili danno, infatti, informazioni simili dal punto di vista della direzionalità: i neuroni molto estesi, infatti, hanno dendriti più lunghi, e questi sono solitamente orientati lungo una direzione preferenziale. Stesso discorso vale per la lunghezza dell'asse principale: infatti, se l'estensione neuronale è maggiore, sarà maggiore anche la lunghezza di questo. Infine, più la cellula è direzionale, più è significativo conoscerne l'angolo del cono in chi essa è racchiusa, poiché più la cellula tende a svilupparsi lungo una direzione preferenziale, più il valore dell'angolo del cono diminuisce con il tempo.
2. Cluster 2 (Dimensione frattale, Area del soma, Coefficiente di regressione): queste variabili morfologiche sono in una posizione intermedia nel plot perché sono circa costanti per tutte le cellule, quindi non sono significative se analizzate dal punto di vista della direzionalità.
3. Cluster 3: (Numero massimo di intersezioni, Raggio critico).

Per quanto riguarda l'asse y, si possono raggruppare le variabili in altri tre *cluster*:

1. Cluster A (Dimensione Frattale, Coefficiente di regressione, Numero massimo di intersezioni, Raggio critico). Queste variabili sono legate alla complessità dell'albero dendritico: la dimensione frattale è una misura della complessità della cellula per sua definizione. Il coefficiente di regressione, di contro, dà informazioni sulla complessità della cellula nello spazio: è indice, infatti, della variazione del numero di intersezioni man mano che ci si allontana dal centro del soma. Il numero massimo di intersezioni è un indice del numero massimo di ramificazioni dei dendriti: più complessa è la cellula, maggiori saranno le ramificazioni. Il raggio critico, infine, è strettamente legato a Nm: infatti indica la distanza dal centro del soma a cui ho il massimo numero di ramificazioni;
2. Cluster B (Area del soma, Percorso sull'asse principale, Estensione);

3. Cluster C (Angolo del cono). Questa variabile influisce in maniera inversa: una cellula con un orientamento preferenziale molto marcato, infatti, presenta un'arborizzazione meno complessa di una cellula che si estende su 360°.

Per ulteriori informazioni, si rimanda al punto [79] della bibliografia.

► Grafico delle cellule

Si riporta in Fig. 5 il grafico degli oggetti, in questo caso delle cellule.

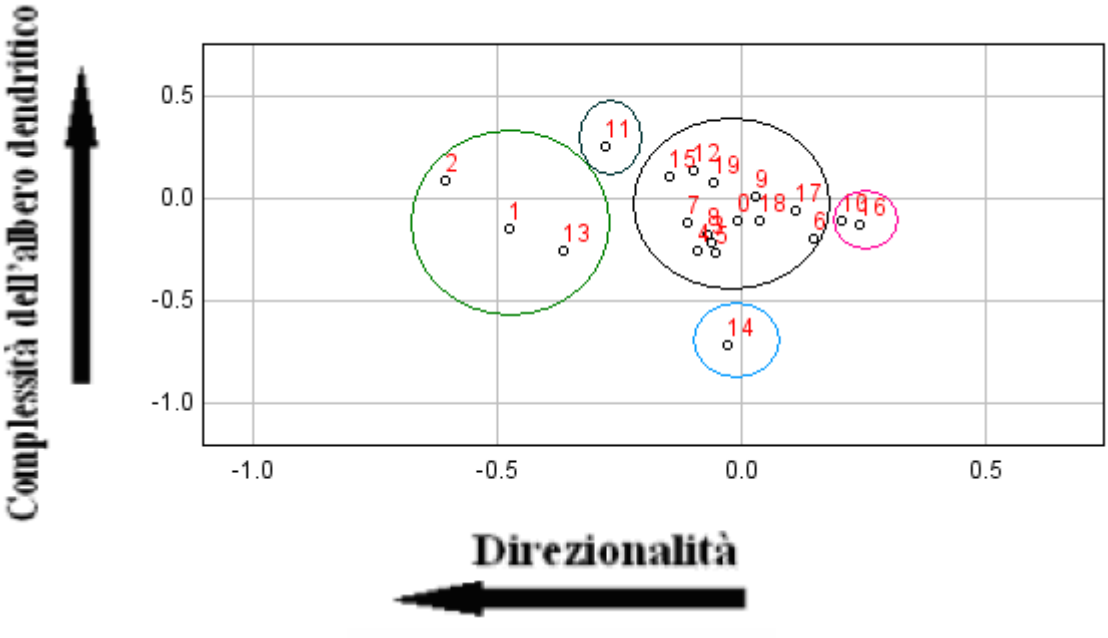


Fig. 5 Grafico delle cellule interpretato in base al significato degli assi

Dal grafico delle cellule si nota che la maggior parte sono molto vicine tra loro, tuttavia ci sono alcune cellule che discostano dal gruppo, cioè hanno direzionalità e complessità diverse dalla media.

Osservando le foto delle cellule è evidente come effettivamente esse vengono disposte nel grafico in base alle loro caratteristiche morfologiche. Per esempi di foto di cellule, si rimanda al punto [79] della bibliografia.

► Grafico delle condizioni

Infine, osservando l'ultimo grafico, cioè quello relativo alle condizioni, ovvero i giorni, si può confermare che l'interpretazione data agli assi è corretta. Lo si riporta in Fig. 6.

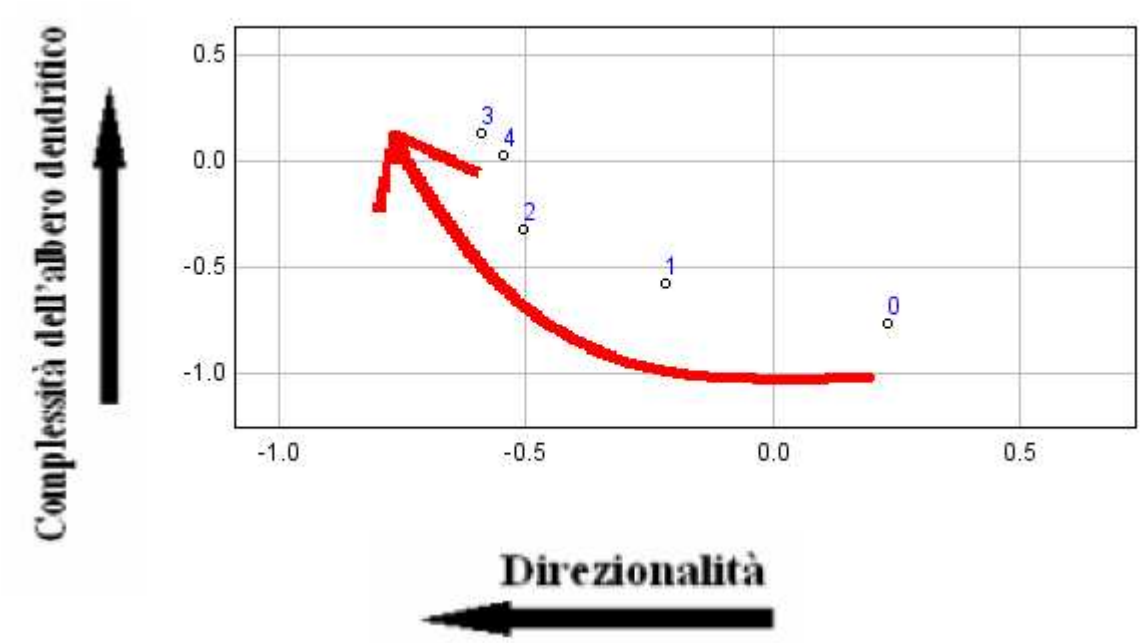


Fig. 6 Grafico delle condizioni interpretato in base al significato dato agli assi

Si può vedere come, all'aumentare dei giorni, aumentino complessità e direzionalità. Infatti, la cellula nel medium di coltura tende a crescere, sviluppando il proprio albero dendritico, che diventa, quindi, più complesso. Inoltre, la maggior parte delle cellule tende ad avere una certa direzione preferenziale verso cui si sviluppa.
Nel grafico riportato, ciò è vero per le prime 4 condizioni considerate, mentre nella quinta ho una diminuzione della complessità: ciò potrebbe essere dovuto al fatto che la cellula inizia a morire, e parte dei dendriti si deteriorano.

4.1.2.2 Cellule wild-type e di modello patologico

Come già ribadito nei precedenti paragrafi, si è utilizzato un secondo datamatrix, ottenuto durante il suo lavoro di tesi triennale da Margherita Brancadoro, per testare ulteriormente la validità del plug-

in implementato in Java. Le colture da cui si sono fotografate le cellule ed estratti i parametri morfologici necessari all'analisi contenevano cellule del Purkinje wild-type, ossia cellule estratte dal cervelletto di topi En2+/+, e cellule del Purkinje estratte da En2+/-.

Uno dei modelli rappresentativi di alcune anomalie presenti nell'autismo è infatti l'*Engrailed 2* (En2). Esso è portatore di un vasto insieme di caratteristiche, presenta un deficit nel numero di cellule del Purkinje e di quelle granulari, e riporta anormalità a livello della foliazione cerebellare: questi sono tutti requisiti propri della sindrome di autismo. I modelli Engrailed mantengono, inoltre, la diversità tra i sessi che si riscontra negli esseri umani: il numero di femmine autistiche è, infatti, minore di quello dei maschi.

Il datamatrix ottenuto ha dimensioni 24*8: infatti, le condizioni scelte sono i giorni comuni a tutte le cellule En2+/+ ed En2+/-, e sono due (divisione 10 e divisione 15). Inoltre, le cellule utilizzate sono 7 per i topi wild-type e 5 per i topi modelli animali dell'autismo, per un totale di 12. Le variabili morfologiche sono le stesse riportate già nel paragrafo 4.1.1.

Si riportano, come in precedenza, le tre coppie di grafici, ottenuti con gli algoritmi implementati in Matlab e in Java (Fig. 7-9). Si interpreteranno solo successivamente uno alla volta.

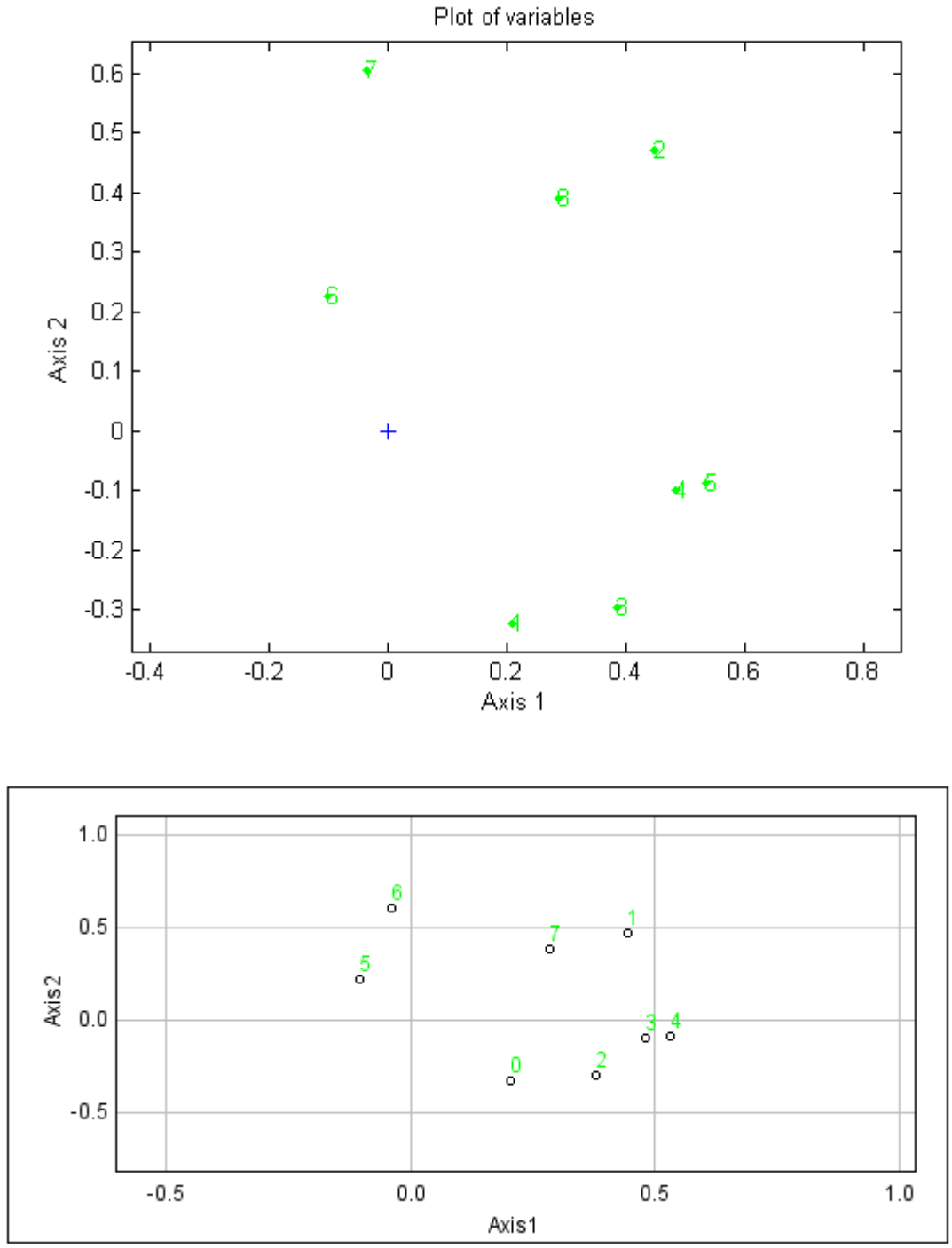


Fig. 7 Grafico delle variabili

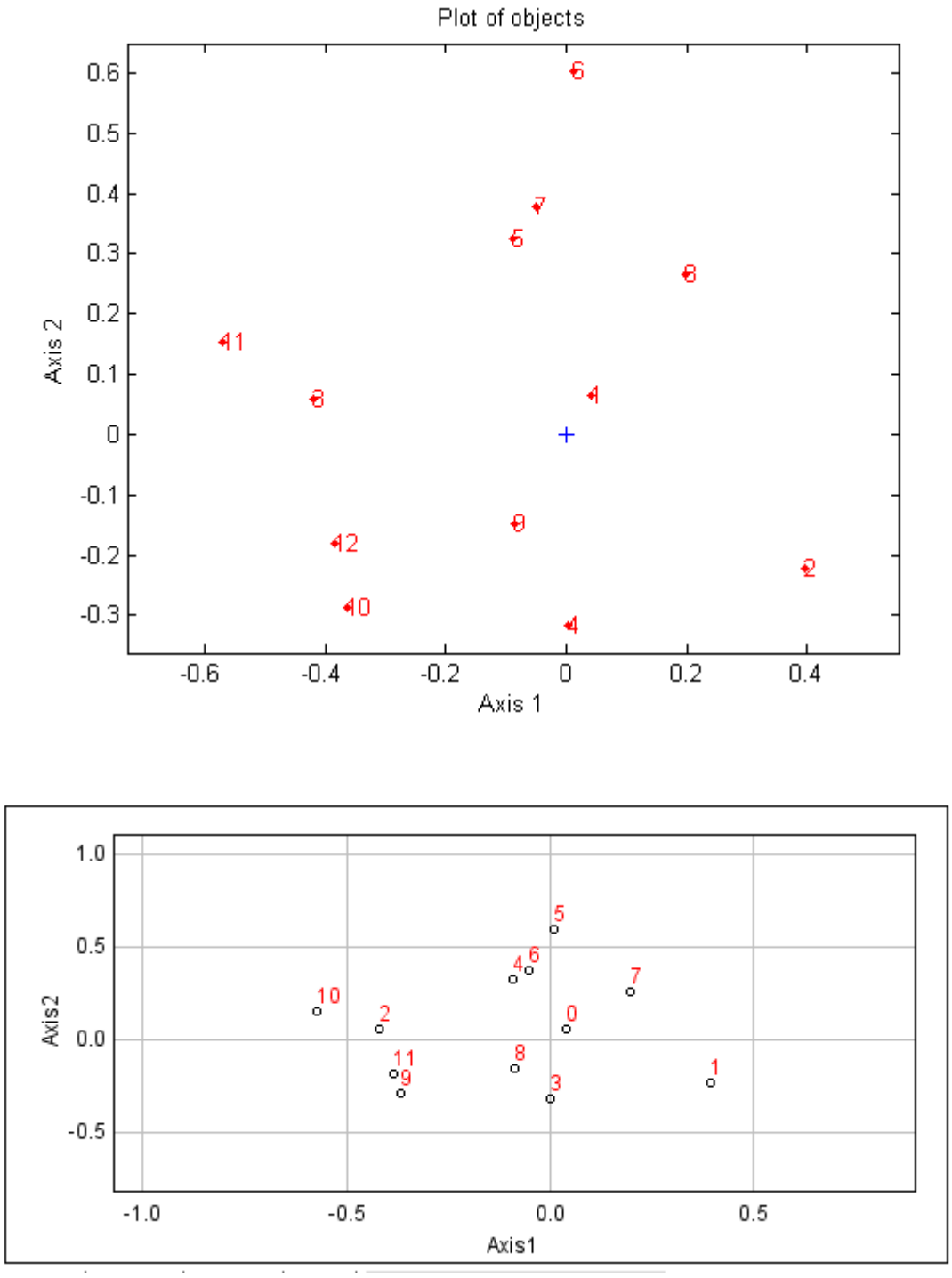


Fig. 8 Grafico delle cellule (oggetti)

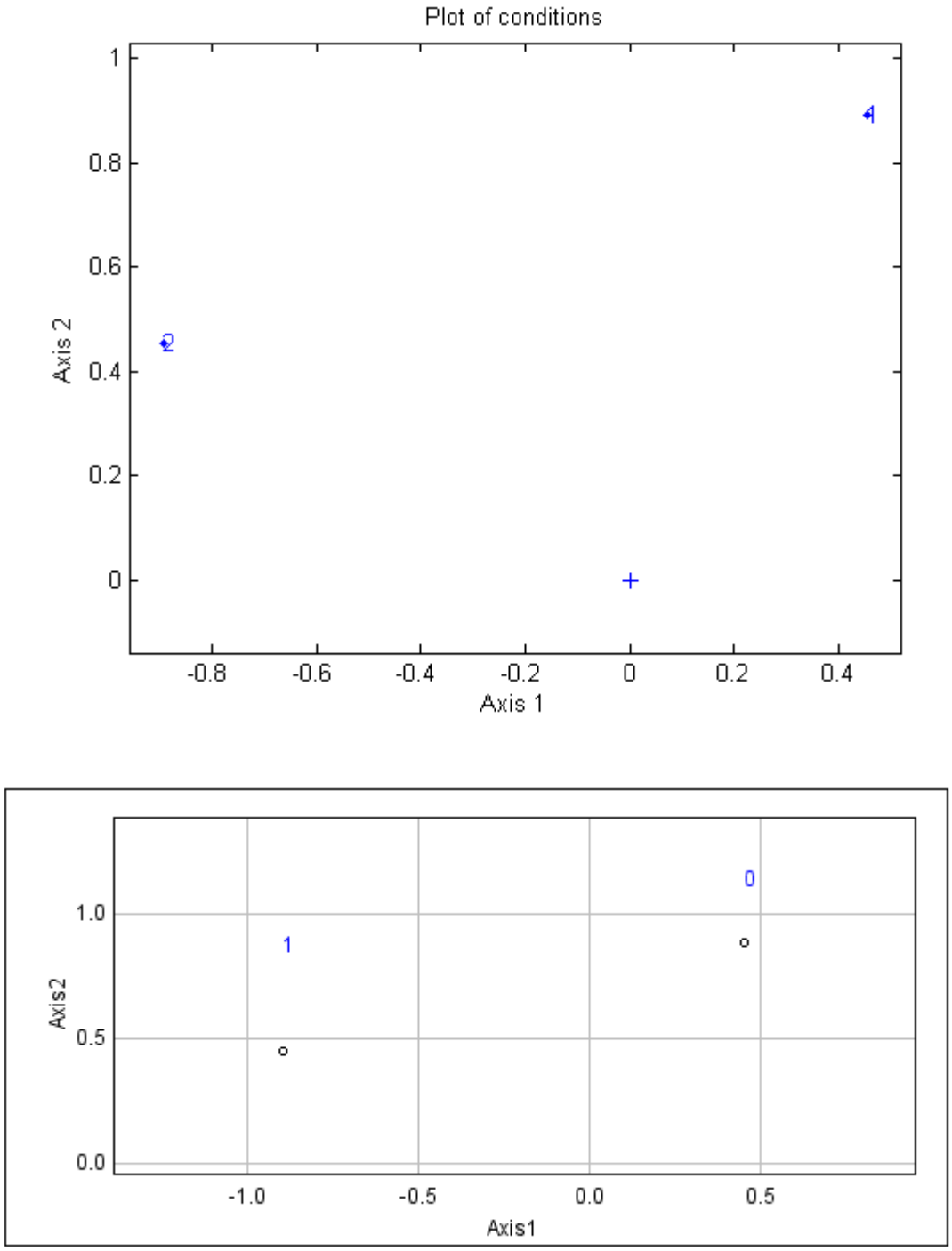


Fig. 9 Grafico delle divisioni (condizioni)

Anche in questo caso, i grafici danno la stessa informazione, siano essi ottenuti con l'editor Matlab o con il plug-in in Java, a riprova della corretta implementazione di quest'ultimo.

Per dare un senso ai grafici, è stato necessario dare un'interpretazione agli assi.

L’asse x rappresenta la direzionalità con verso crescente da sinistra a destra, mentre l’asse y rappresenta la complessità dell’albero dendritico con verso crescente dal basso verso l’alto.
Questa interpretazione è giustificata analizzando ciascuno dei tre grafici.

► Grafico delle variabili

In Fig. 10 si riporta il plot delle variabili con la relativa interpretazione data agli assi.

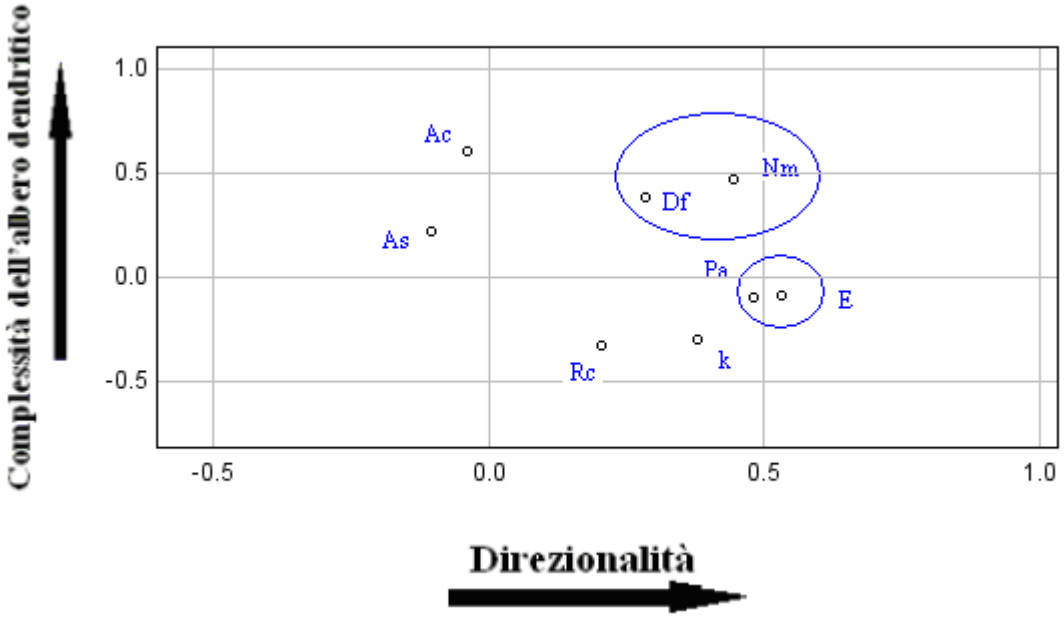


Fig. 10 Plot delle variabili interpretato in base al significato degli assi

Dal grafico delle variabili si osserva che esse sono distribuite sia lungo l’asse x che lungo l’asse y.
All’interno dei cerchi blu sono racchiuse le variabili per le quali emerge un risultato significativo dal grafico.
In basso a destra si trovano l’Estensione (E) e il Percorso sull’asse (Pa). La prima variabile fornisce un’indicazione sulla direzionalità: i dendriti hanno, infatti, una maggiore lunghezza se sono allineati lungo una direzione preferenziale. Lo stesso discorso vale per l’asse principale, che avrà una maggiore lunghezza e quindi il percorso avrà

un valore maggiore: una cellula molto direzionale tende ad estendersi lungo la direzione dell'asse principale.

Al contrario, l'angolo del cono e l'area del soma forniscono un'indicazione opposta, infatti un angolo ampio e un soma grande sono indice di una cellula poco orientata.

La posizione in alto all'interno del grafico della Dimensione frattale (Df) è data dal fatto che tale variabile è indice della complessità della cellula.

Lo stesso discorso vale per il Numero massimo di intersezioni (Nm) in quanto maggiore è il numero di ramificazione e maggiore è la complessità della cellula.

► Grafico delle cellule

In Fig. 11 si riporta il secondo grafico, quello relativo agli oggetti, in questo caso le cellule.

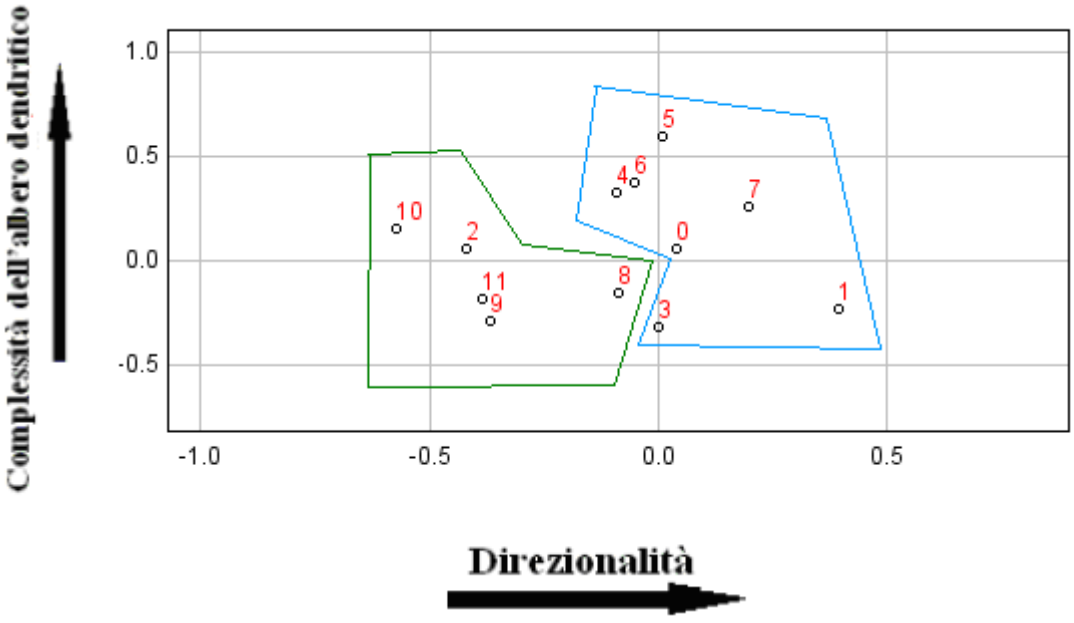


Fig. 11 Grafico delle cellule interpretato in base al significato degli assi

Le cellule contrassegnate con i numeri che vanno dallo 0 al 6 appartengono alla prima coltura, quindi sono le cellule appartenenti al topo wild-type; mentre le altre, dal numero 9 all' 11, sono cellule di topo En2 +/-.

È stato possibile suddividere le cellule in due gruppi, che corrispondono alle due diverse colture. Si può osservare come le cellule En2 +/- (area racchiusa nel poligono verde in Fig. 11) presentano una bassa complessità dell'albero dendritico e una direzionalità contenuta rispetto a quelle di topo En2 +/+ (area racchiusa nel poligono blu in Fig.118). Ciò è concorde a quanto osservato nell'analisi morfologica.

► Grafico delle condizioni

Il grafico dei giorni (Fig. 12) rappresenta le condizioni utilizzate per eseguire l'analisi. Come già sottolineato, si avevano a disposizione solo due tempi in comune ad entrambi i tipi di cellule: il giorno 10 e il giorno 15.

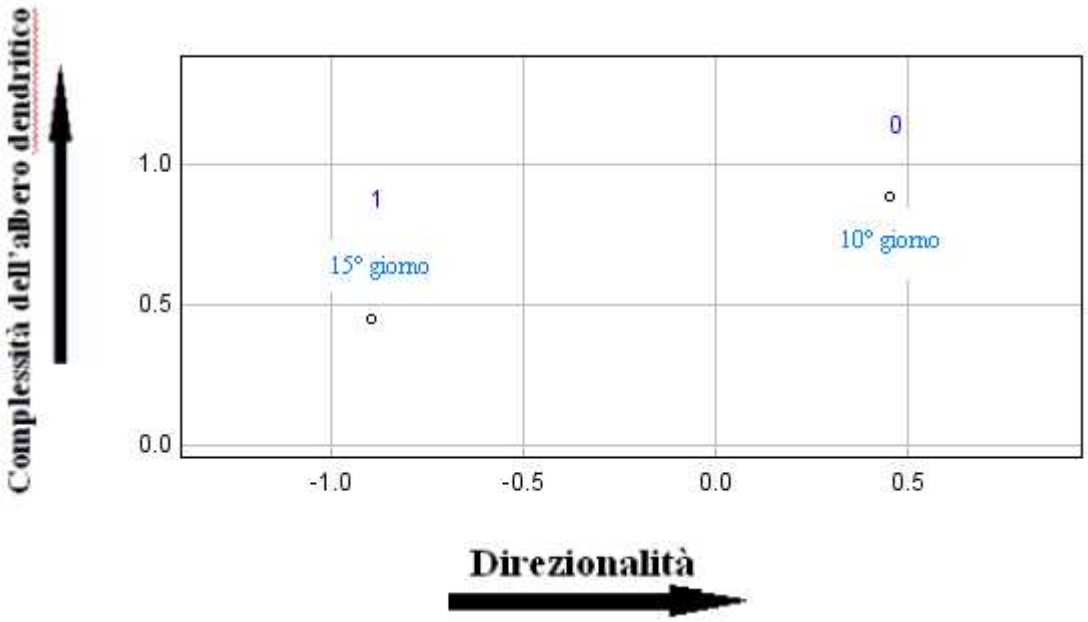


Fig. 12 Grafico delle condizioni interpretato in base al significato degli assi

Osservando il grafico si può affermare che passando dal giorno 10 al 15 diminuiscono l'estensione, la complessità e le dimensioni, questo perché le cellule iniziano a soffrire prima di andare incontro alla morte. Ciò si verifica, però, solo per le cellule En2 +/- e non per quelle di topo wild-type.

4.1.2.3 Cellule provenienti da claustro

Per quanto riguarda il claustro, si sono acquisite delle immagini da vetrini fissati.

Questi sono stati preparati dai collaboratori della prof.ssa Maura Castagna, del Laboratorio di Anatomia Patologica della Facoltà di Medicina e Chirurgia dell'Università di Pisa.

Il materiale utilizzato per ottenere i vetrini è autoptico, proveniente da cervelli di individui sani.

I vetrini sono stati ottenuti fissando slice, di diverso spessore, di claustro.

I vetrini contengono slice di claustro, regione della sostanza grigia cerebrale, di spessore variabile: in particolare, 10 μm, 20 μm e 30 μm. Le acquisizioni sono state eseguite a 40X, e non a 20X come negli scorsi lavori di tesi. A 20X, infatti, il rumore di background risultava essere maggiore rispetto alle acquisizioni a 40X, in cui invece le cellule appaiono scure, rispetto al background, più chiaro con poco rumore, come mostrato in Fig. 13

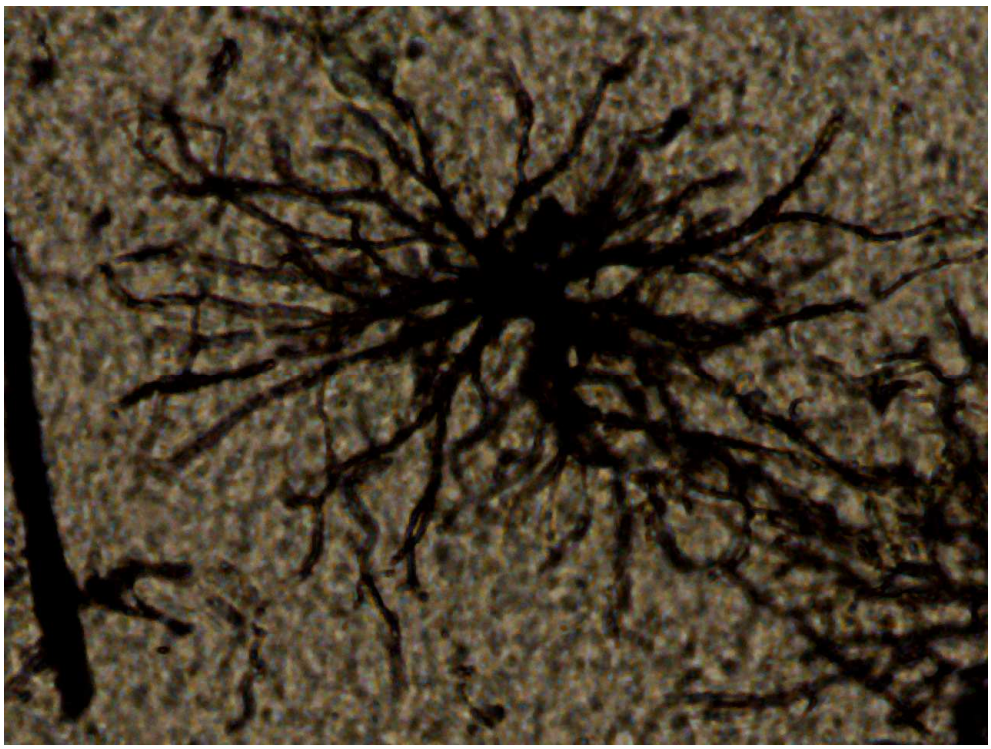


Fig. 13 Esempio di neurone acquisito a 40X in una slice di profondità 30 μm

Il problema dell'acquisizione a ingrandimenti così elevati è che non sempre è possibile racchiudere nel campo visivo della fotocamera collegata al microscopio l'intera cellula: è necessario, quindi, acquisire il neurone in più immagini, e poi fonderle insieme in modo da ottenerne una sola, idonea all'elaborazione e alla successiva analisi morfologica.

Per fare questo, ci si è avvalsi di un particolare tool interattivo di Matlab, il *Control Point Selection Tool*.

Nell'Appendice C sono riportate le righe di codice implementate per questo tipo di operazione.

Prima di caricare le immagini nel tool, si è operato un miglioramento del contrasto, dove necessario, e, per migliorare le prestazioni computazionali, si è processata l'immagine in modo da ottenerne una in livelli di grigio: ciò permette di avere un'immagine rappresentata da una matrice bidimensionale e non tridimensionale, diminuendo i tempi di elaborazione, ma allo stesso tempo non perdendo informazioni utili: l'obiettivo, infatti, è sempre ottenere lo scheletro del neurone, che è un'immagine binaria.

Ritornando al *Control Point Selection Tool*, esso permette di caricare all'interno di una GUI, richiamata sul workspace di Matlab attraverso uno specifico comando (Fig. 14)

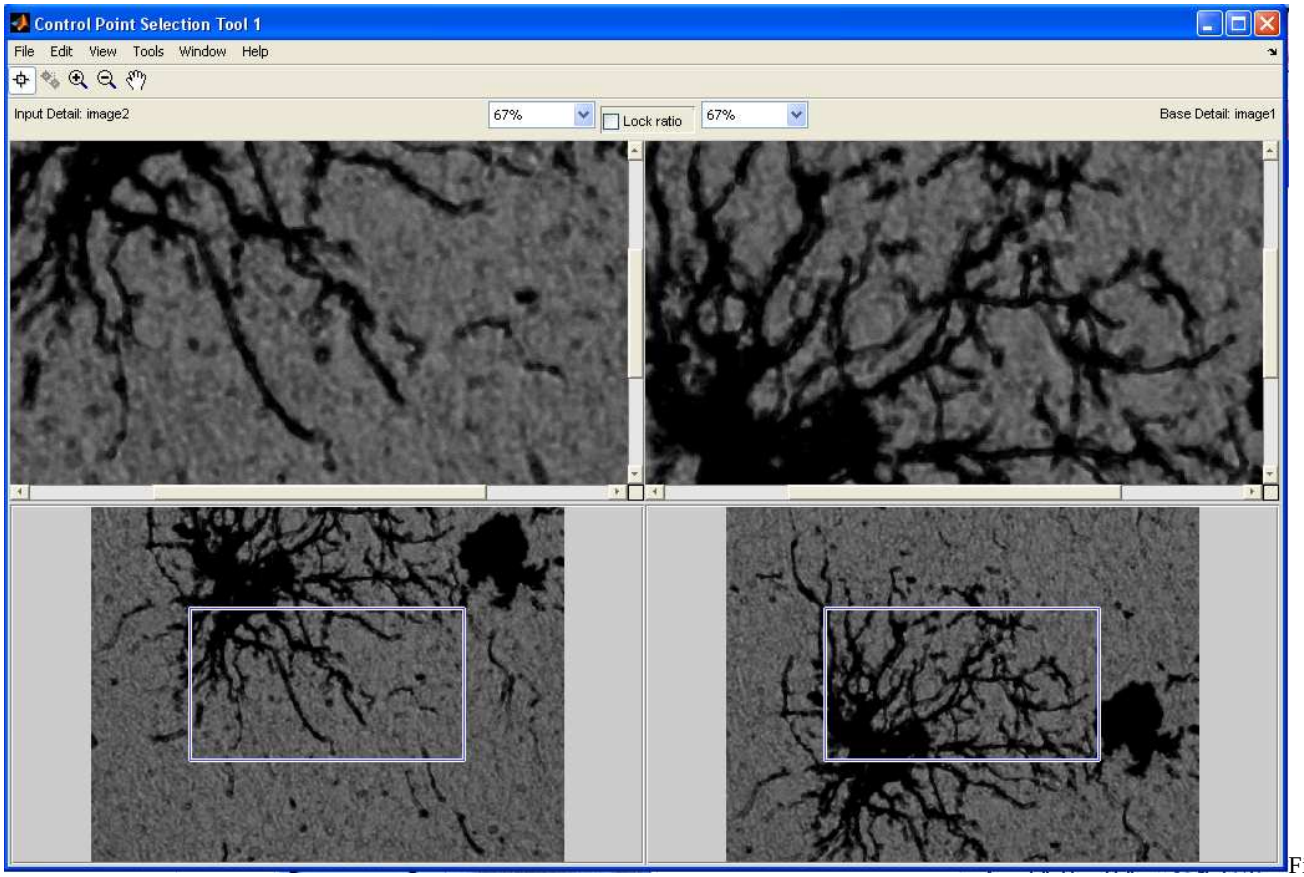


Fig.

Fig. 14 Schermata del Control Point Selection Tool

Le due immagini sono visualizzate nella loro interezza in basso, e su di esse è presente un riquadro mobile: in questo modo, l'utente può selezionare una zona del neurone da ingrandire; tale ingrandimento è visibile nella GUI stessa, in alto. Nei riquadri in cui le immagini sono zoomate, l'utente può, cliccando con il mouse, individuare dei repere, o *landmark*. Questi sono dei punti che si utilizzeranno come riferimento: infatti, a coppie, questi identificano una stessa zona anatomica neuronale. Il numero di landmark da selezionare è funzione della trasformazione che si vuole compiere per ottenere la registrazione. Infatti, è possibile implementare, attraverso opportuni parametri previsti nella sintassi Matlab, diversi tipi di trasformazioni che permettono la registrazione di due immagini. In questo caso, il microscopio non distorce in modo significativo le immagini, quindi è possibile utilizzare per la "fusione" una delle più semplici trasformazioni globali. Per questo tipo di trasformazione, sono necessarie almeno due coppie landmark, ma per aumentare la precisione nella registrazione se ne sono individuate almeno tre

coppie. Completata la ricerca dei *repere*, attraverso la GUI se ne esportano sul workspace di Matlab le coordinate x e y di ciascun punto, organizzate in due matrici, una per ogni immagine.

A questo punto, si può procedere con l'analisi morfologica, attraverso l'interfaccia grafica presente all'interno di Ne.Mo., ottenendo per ogni cellula dei datamatrix di dimensioni 1*177.
Si è successivamente organizzato il datamatrix complessivo, in modo da renderlo idoneo all'elaborazione con la tecnica di analisi multivariata Tucker3.
In primo luogo, si sono estratte le otto variabili, dalle 177 ottenute dall'analisi morfologica (vedi paragrafo 4.1.1).
Si è notato, successivamente, che all'interno del claustro ci sono due diversi tipi di cellule: si possono notare neuroni che si sviluppano lungo una direzione preferenziale (Fig. 15)

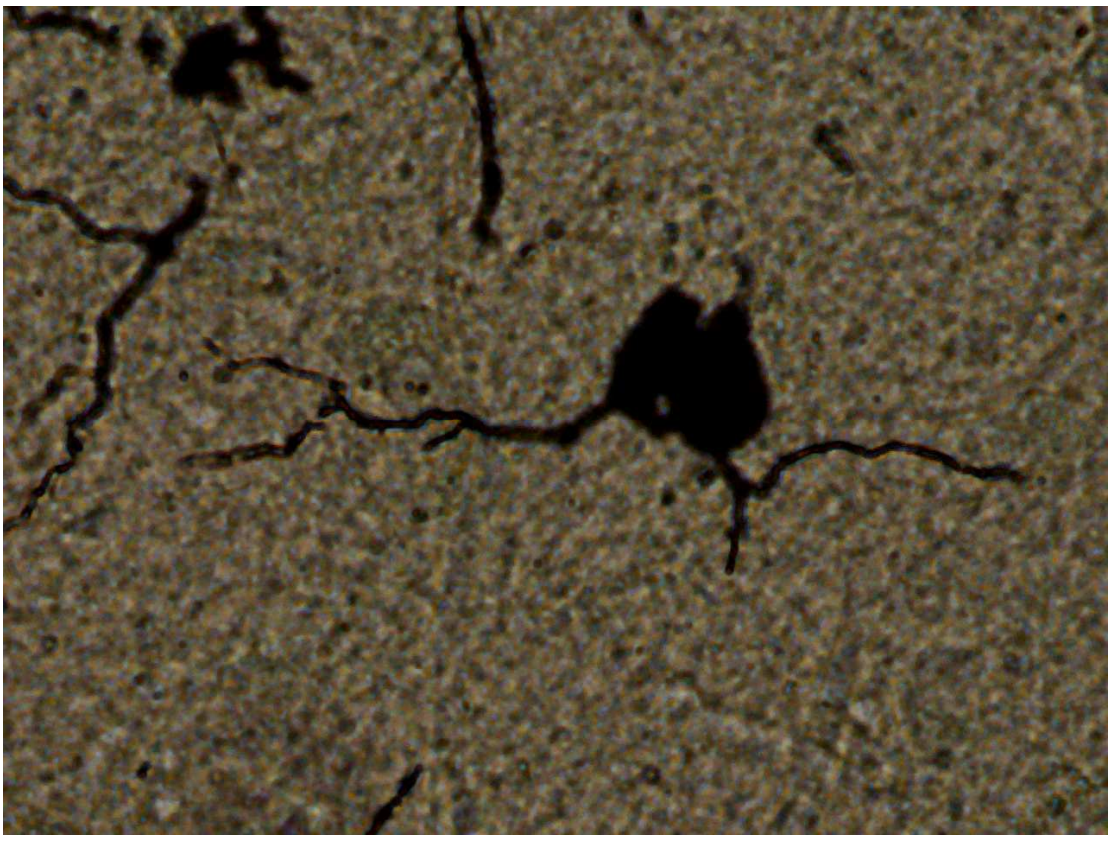


Fig. 15 Neuroni che si estendono lungo una direzione preferenziale

e neuroni che invece si sviluppano a 360° (Fig. 16)

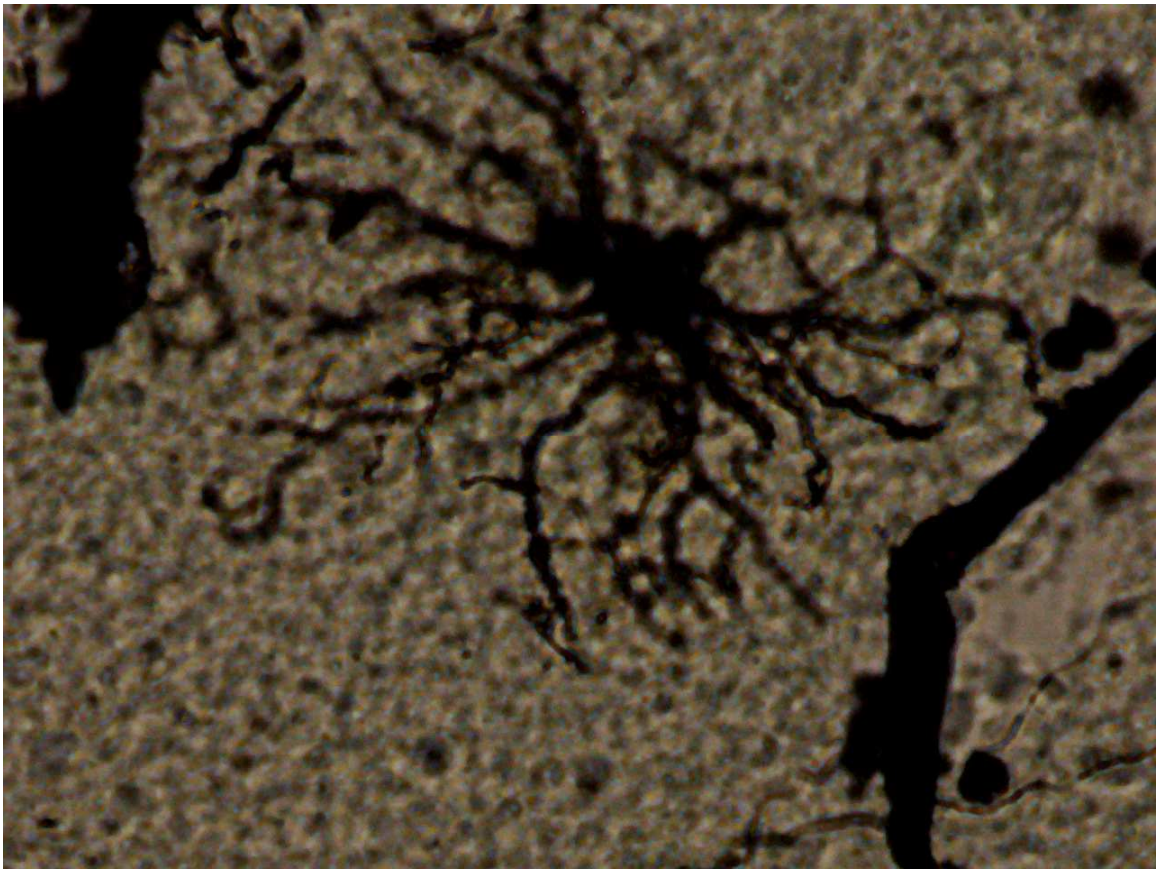


Fig. 16 Esempio di neurone con estensione radiale a 360°

Come già evidenziato, i neuroni sono fotografati da vetrini contenenti slice di claustro con diverso spessore, in particolare 10 μm, 20 μm e 30 μm. Si è quindi pensato, per ogni spessore, di prendere tre neuroni del primo tipo e tre del secondo, per un totale di 6 neuroni: questi rappresenteranno, all'interno del datamatrix, gli oggetti.

Il numero esiguo di cellule è dovuto al fatto che non si avevano a disposizione tanti vetrini su cui fotografare i neuroni.

Come condizione si sono considerati i tre diversi spessori delle *slice* di claustro, da cui sono stati fotografati i due tipi neuronali.

Il datamatrix finale, quindi, avrà dimensioni (3*6)*8.

L'obiettivo di quest'analisi è dimostrare che la tecnica Tucker3, applicata in questo particolare caso, riesce a visualizzare le differenze che ci sono tra i due tipi neuronali individuati. Si analizzerà, quindi, solo il plot degli oggetti, essendo questo il più significativo tra i grafici.

Il risultato è visibile in Fig. 17: anche in questo caso, non c'è differenza nei risultati ottenuti con l'editor Matlab o con il plug-in.

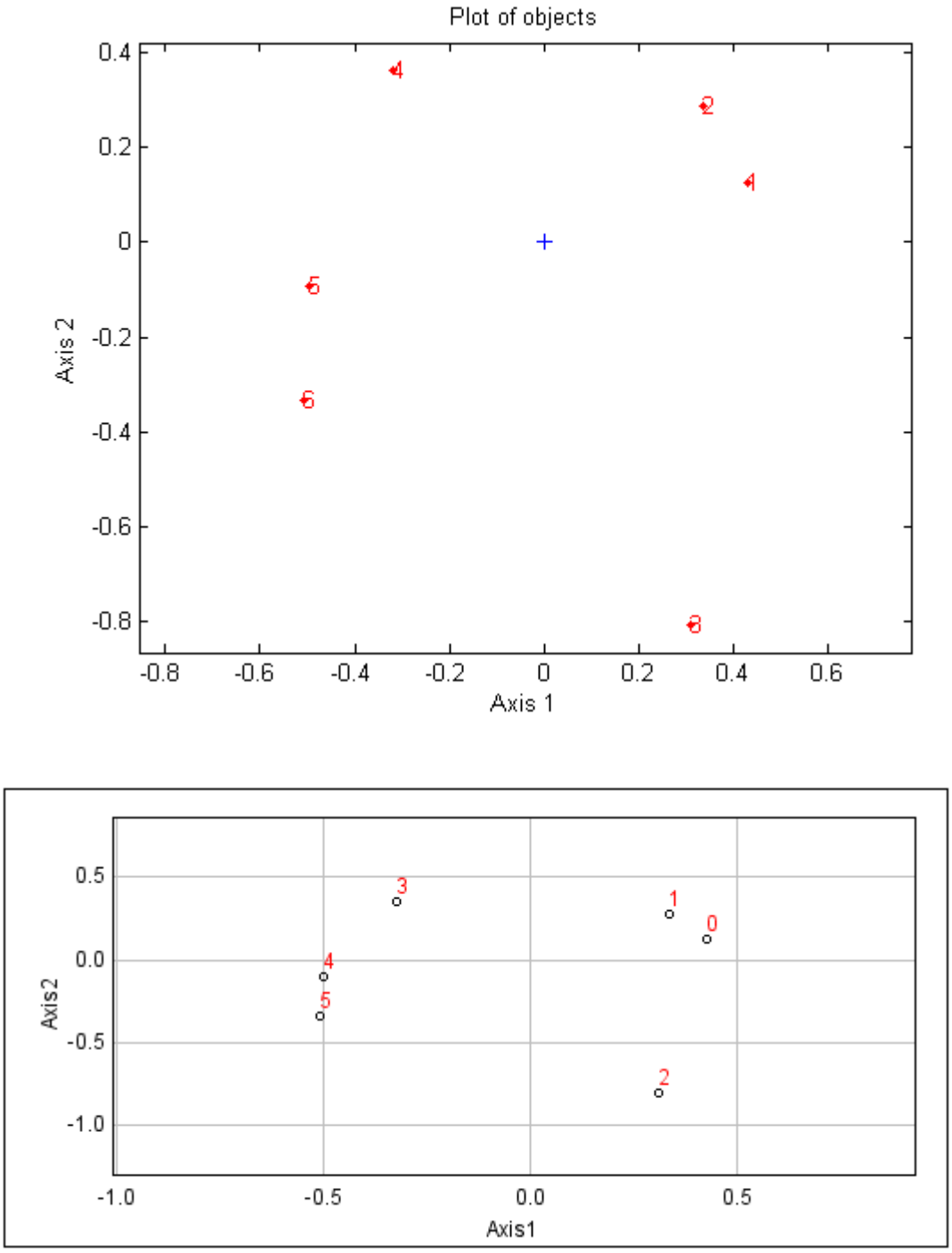


Fig. 17 Grafico degli oggetti

Come nei casi discussi, è necessario dare un significato agli assi, per poter poi interpretare il grafico: in questo caso, l'asse delle x

rappresenta la direzionalità con verso crescente da destra verso sinistra, mentre l'asse y rappresenta la complessità dell'albero dendritico, con verso crescente dal basso verso l'alto. A questo punto, è possibile analizzare il plot degli oggetti, che giustificherà la scelta dell'interpretazione degli assi.

► Grafico delle cellule

Si riporta in Fig.18 il grafico degli oggetti, con la relativa interpretazione data agli assi.

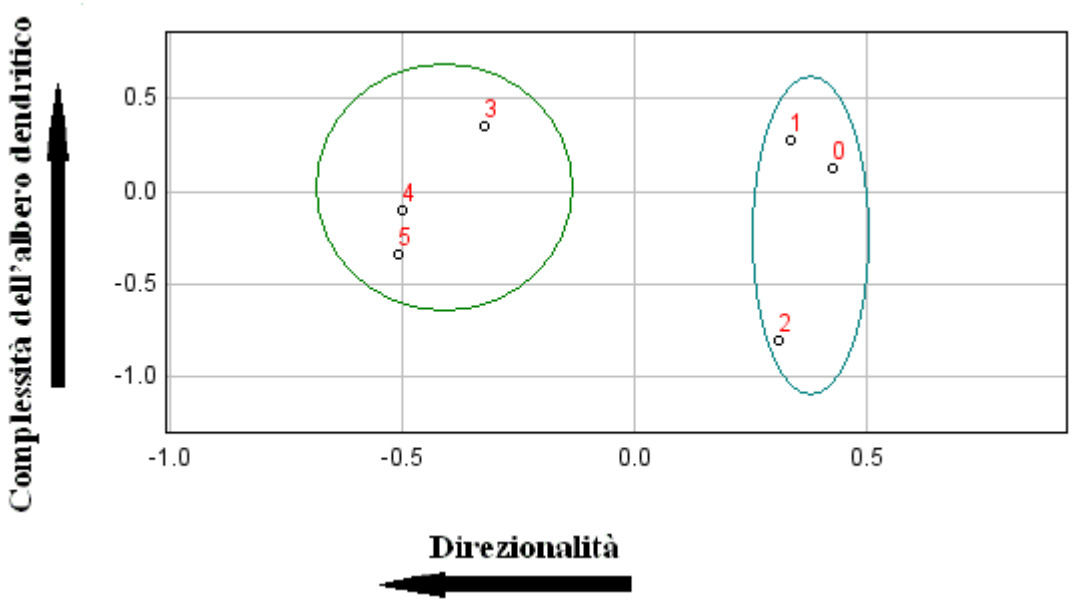


Fig. 18 Grafico degli oggetti interpretato in base al significato degli assi

Osservando il grafico, si possono effettivamente dividere gli oggetti in due cluster:

1. Cluster 1 (verde): raggruppa le cellule più direzionali;
2. Cluster 2 (blu): raggruppa le cellule che si estendono a 360°, quindi meno direzionali.

All'interno dei due cluster, è possibile individuare cellule con complessità variabile. Ad esempio, la cellula contrassegnata dal numero "2" (Fig. 19) è effettivamente una cellula poco direzionale, e che, inoltre, presenta poca complessità nella struttura morfologica dell'albero dendritico.

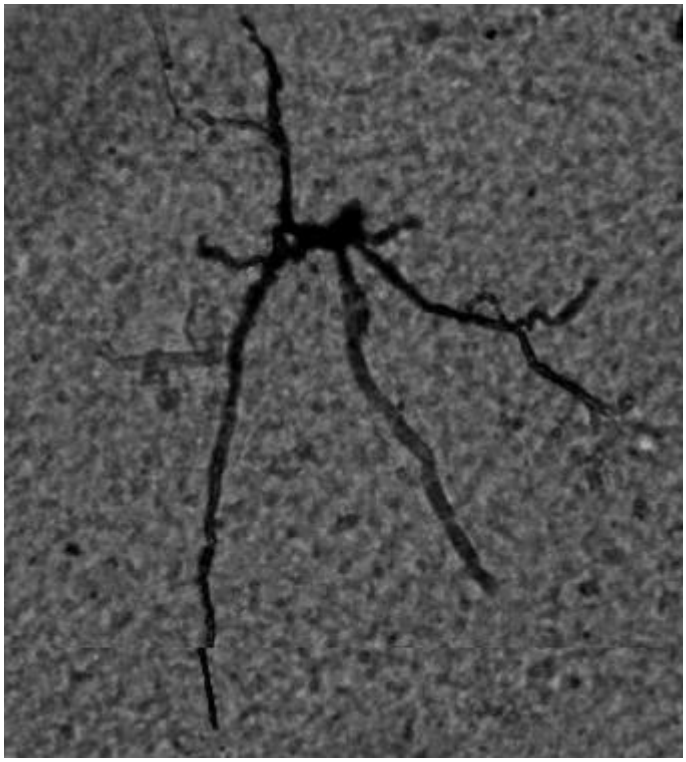


Fig. 19 Esempio di cellula (cellula no.2)

In definitiva, dunque, il metodo Tucker3 riesce nella classificazione di diversi tipi cellulari, e all'interno dei vari cluster, rileva ulteriori differenze.

4.2 Analisi multivariata su test di tossicità

Per evidenziare ancora una volta che la tecnica Tucker3 può essere utilizzata nei campi di ricerca più disparati, si sono analizzati dati estratti da test di tossicità.

Infatti, all'interno di un progetto di dottorato in Morfologia e Funzione normale e patologica di cellule e tessuti, sono stati condotti dalla Dott.ssa Nadia Ucciferri degli esperimenti su tre diversi tipi cellulari:

1. C3A, linea cellulare di epatociti;
2. HUVEc (Human Umbilical Vein Endothelial Cells), cellule epiteliali;
3. CACO-2, linea cellulare di epiteliali intestinali umane.

Per ogni coltura di questi tre tipi cellulari si è valutata la tossicità per un tipo di nanoparticella, somministrata in batterie di dieci concentrazioni.

La tossicità è stata valutata a livello di vitalità (con i test *Alamar Blue*, indice dell'attività metabolica, e LDH, che invece valuta l'integrità di membrana) e a livello sub letale tipo infiammazione (IL8, ICAM), stress ossidativo (GSH), funzionalità (vWF per HUVEc, Albumina e phalloidina per C3A e Trans Epithelial Resistance per CACO-2).

Per il momento, sono disponibili i dati per un'unica nanoparticella, Ag NM 300, e, per quanto riguarda i test, sono state effettuate solo valutazioni della tossicità a livello di vitalità. I test sono stati ripetuti, per ogni concentrazione di nanoparticella, almeno tre volte: si è quindi effettuata una media degli esiti dei test.

Si è, quindi, costruito il datamatrix secondo le regole già discusse precedentemente, considerando come variabili la batteria di 10 concentrazioni di nanoparticella somministrata alle 3 colture cellulari, che invece rappresentano gli oggetti. Le condizioni sono i 2 test effettuati per ciascuna concentrazione: Alamar Blue e LDH.

Il datamatrix finale, dunque, è di dimensioni (2*3)*10.

Si riportano le tre coppie di grafici, rispettivamente plot di oggetti, variabili e condizioni risultanti dall'avvio dell'editor di Leardi e del plug-in in Java (Fig.20-22).

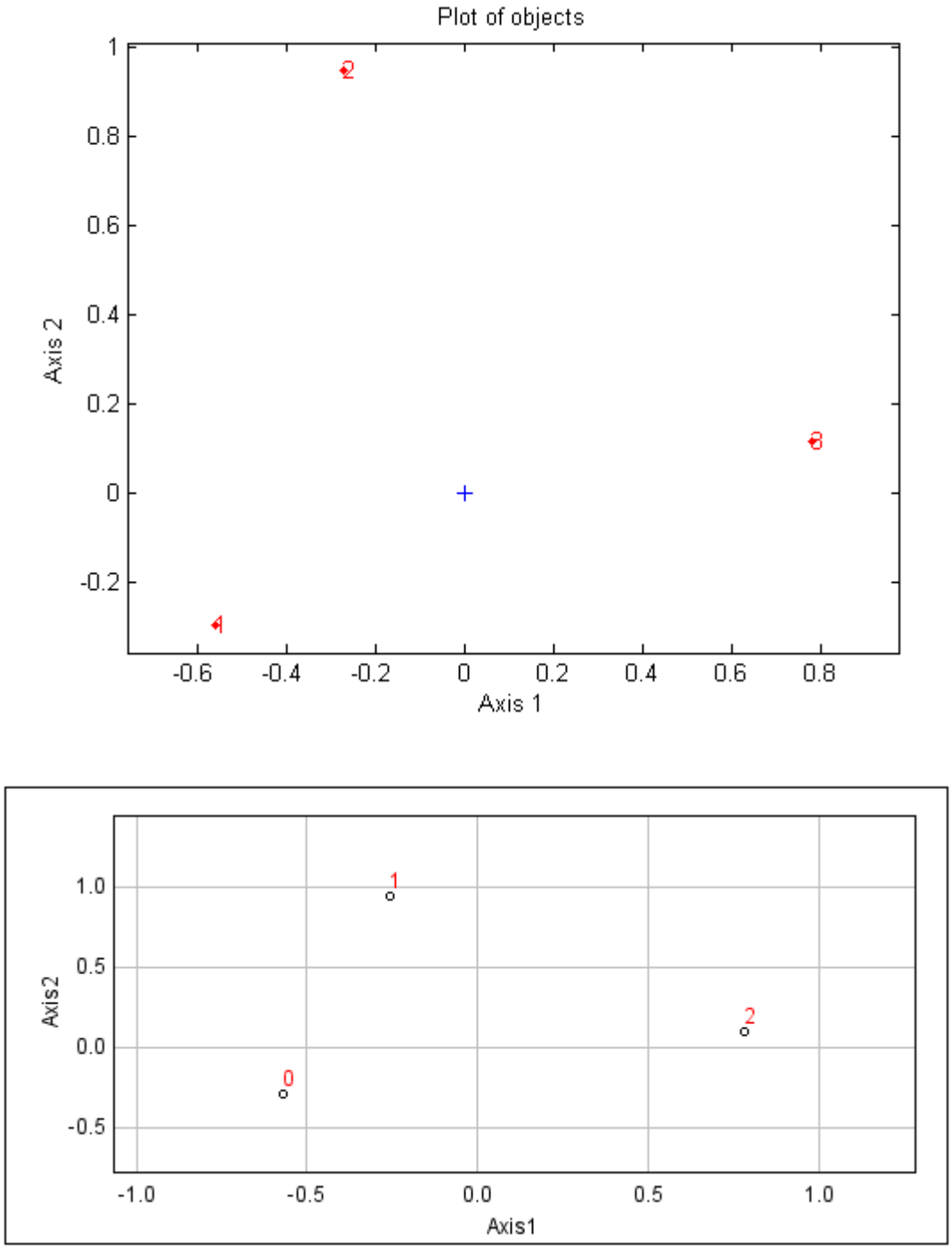


Fig. 20 Grafico degli oggetti

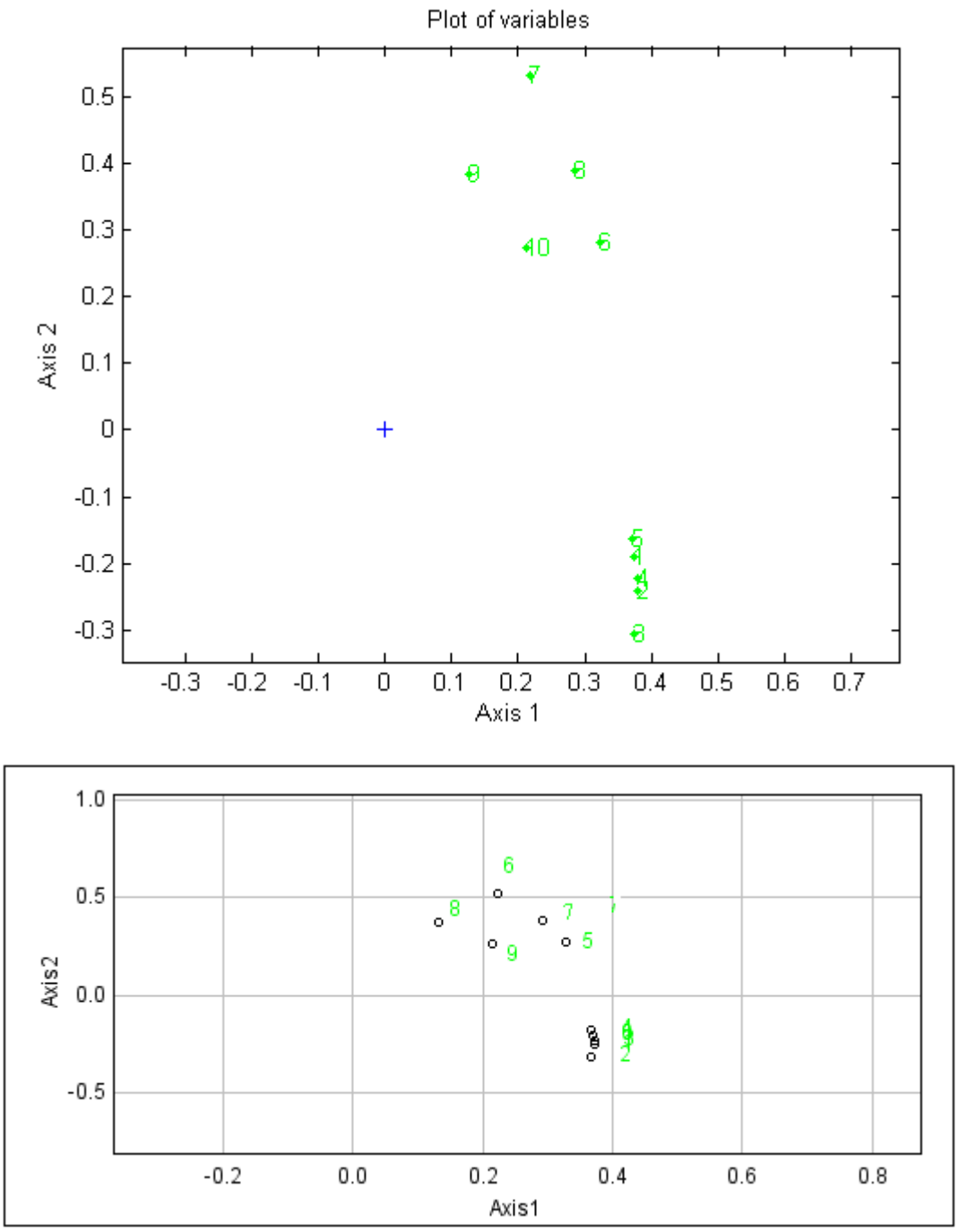


Fig. 21 Grafico delle variabili

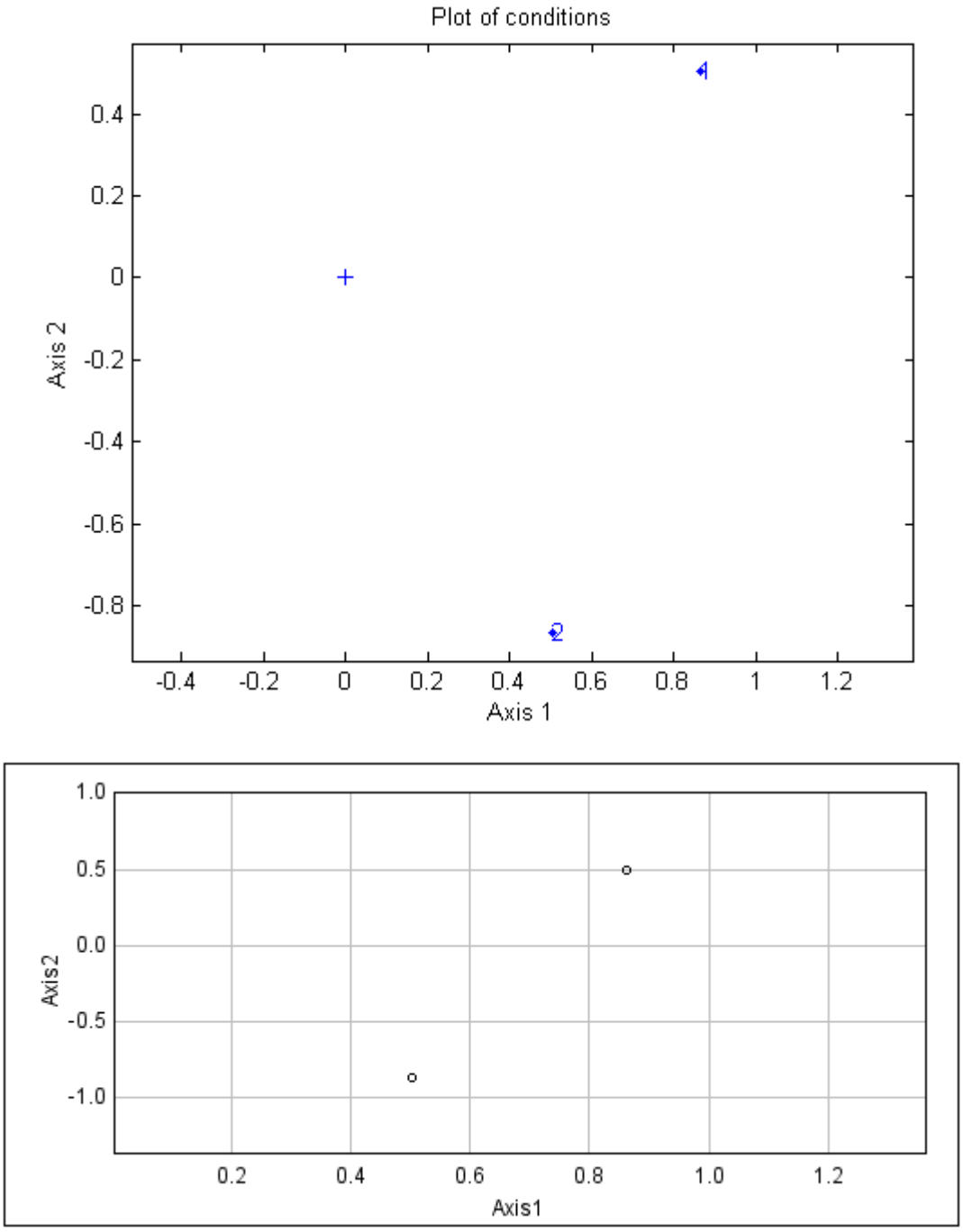


Fig. 22 Grafico delle condizioni

Come nelle precedenti analisi, i grafici ottenuti con l'editor Matlab o con il plug-in Java, risultano essere totalmente equivalenti.

E' necessario dare un'interpretazione agli assi, come si è già fatto per i precedenti casi; la scelta del significato dato sarà più chiara quando, in seguito, si interpreteranno i grafici singolarmente.

L'asse x è un indice della "resistenza" della cellula ad essere attaccata dalla nanoparticella, con direzione che va da sinistra verso

destra; l'asse y, invece, rappresenta la vitalità della cellula, con direzione che va dall'alto verso il basso.

► Grafico delle condizioni

Si riporta il grafico delle condizioni, associando agli assi l'interpretazione enunciata precedentemente (Fig. 23)

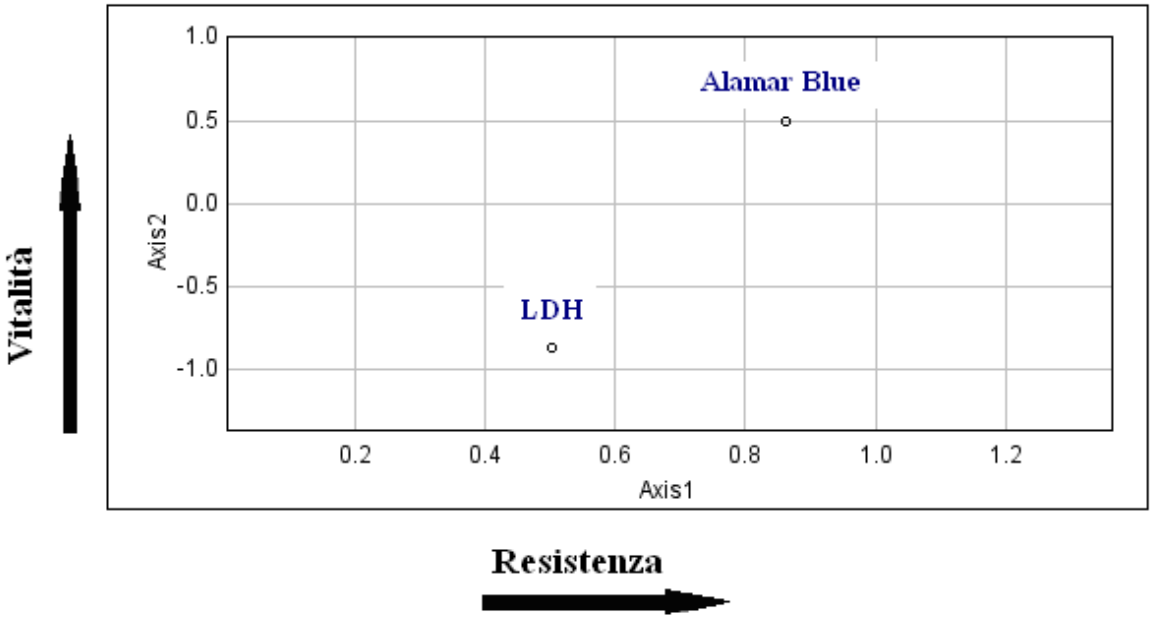


Fig. 23 Grafico delle condizioni, interpretato in base al significato degli assi

Dal grafico si nota che le due condizioni si diversificano soprattutto per quanto riguarda la coordinata y, che rappresenta la vitalità. Infatti i due test riportano informazioni complementari: mentre l'Alamar Blue, infatti, è un indice dell'attività metabolica, direttamente correlata alla vitalità cellulare, il test LDH, essendo una misura della quantità di LDH nel medium di coltura, che aumenta in caso di lisi cellulare, è inversamente correlato alla vitalità stessa.

► Grafico delle variabili

Dal grafico delle variabili (Fig. 24), risulta evidente la netta presenza di due cluster, interpretabili alla luce del significato dato agli assi:

Fig. 24 Grafico delle variabili, interpretato in base al significato dato agli assi

1. Cluster 1 (rosso): individua le concentrazioni di Ag NM300 più basse somministrate alle tre colture cellulari;
2. Cluster 2 (blu): individua le concentrazioni di Ag NM300 più alte somministrate alle tre colture cellulari.

E' intuitivo che la vitalità, in corrispondenza delle concentrazioni minori, è più alta rispetto che alle concentrazioni maggiori di nanoparticella. L'interpretazione dell'asse y è dunque ancora una volta confermata.
Le differenze sull'asse delle x, invece, nei due cluster non sono significative.

► Grafico delle colture cellulari

Si riporta in Fig. 25 il grafico relativo alle tre colture cellulari analizzate.

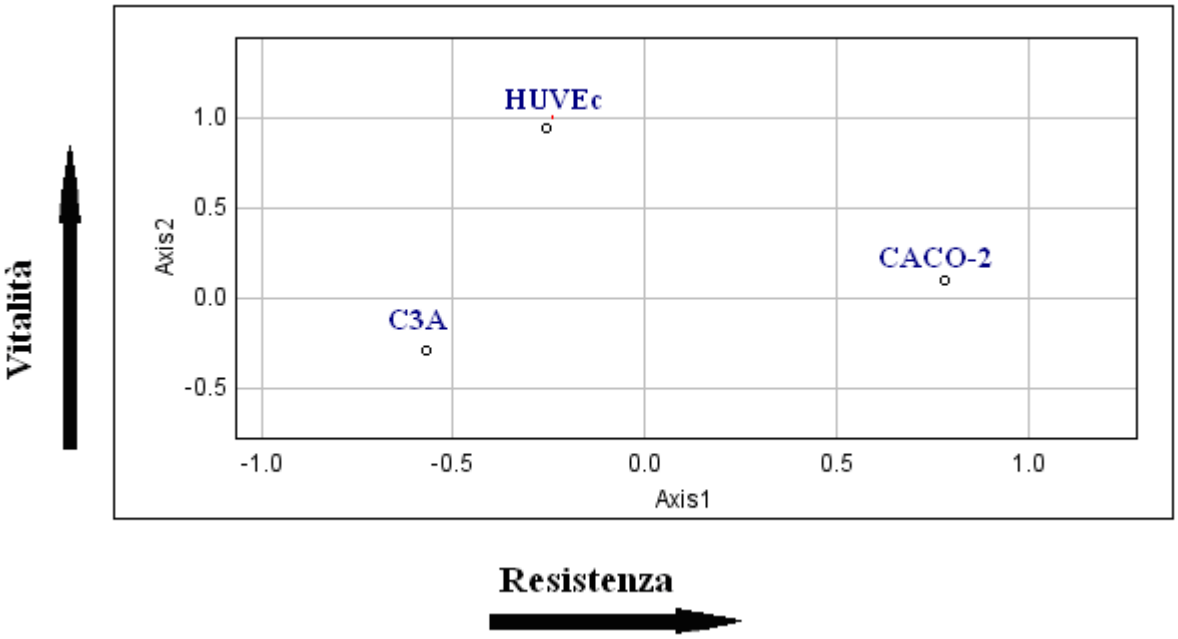


Fig. 25 Grafico degli oggetti, interpretato in base al significato dato agli assi

In questo plot, è comprensibile il significato dato all'asse x.

Infatti, i tre tipi cellulari non esibiscono la stessa reazione alla somministrazione della nanoparticella nel medium di coltura. In particolare, gli epatociti (C3A) sembrano essere meno "resistenti" alla presenza dell'Ag NM300 rispetto alle cellule endoteliali (HUVEc) ed epiteliali intestinali (CACO-2). Ciò è confermato anche in uno studio, condotto da Eva Collnot, su questi tre tipi cellulari [87]. Si riportano, in tabella di Fig. 26, evidenziati attraverso un rettangolo verde, i valori della *Lethal Dose* (LD) per l'Ag NM300 somministrato ai tre tipi cellulari in coltura. In particolare, si riportano i valori ottenuti eseguendo il test Alamar Blue.

		HUVEC	C3A	Caco-2
Alamar blue	55 nm fluoresbrite	625 µg/cm^2	625 µg/cm^2	
	211 nm fluoresbrite	625 µg/cm^2	625 µg/cm^2	
	Au 15 nm	156 µg/cm^2		
	Au 80 nm	156 µg/cm^2		
	NM 101 TiO2	28 µg/cm^2		
	NM300 Ag	25 µg/cm^2	15 µg/cm^2	340 µg/cm^2

Fig. 26 Risultati ottenuti da E. Collnot [87]

LD è un indicatore di letalità: indica la dose necessaria per uccidere il 50% dei soggetti; in questo caso, indica la quantità di nanoparticella da somministrare alle tre colture per far sì che il 50% delle cellule muoiano. I risultati ottenuti rispecchiano quelli ottenuti dall'analisi effettuata tramite 3-way PCA.

E' evidente, dunque, non solo che il plug-in scritto in linguaggio di programmazione Java da risultati del tutto equivalenti a quelli che si ottengono con l'editor Matlab implementato da Leardi, ma anche che la 3-way PCA è un potente strumento per l'analisi di dati multivariati.

Conclusioni

L’obiettivo primario di questo lavoro di tesi è stato l’implementazione di un plug-in, in linguaggio di programmazione Java, che effettuasse una particolare analisi di dati multivariati, la 3-way PCA. Questa permette di analizzare dati che provengono dall’osservazione di un certo numero di variabili su un certo numero di soggetti od oggetti. Rispetto alla classica PCA, la Tucker3, modello di 3-way PCA implementato durante questo lavoro di tesi, permette di analizzare le correlazioni tra le variabili in tre dimensioni: ciò risulta particolarmente utile in campo biomedico, in cui spesso le variabili sono osservate in un certo numero di situazioni, ad esempio in diversi istanti di tempo.

L’applicazione della 3-way PCA di maggiore interesse in questo lavoro di tesi è stata l’analisi di parametri morfologici di neuroni.

Nella prima fase del lavoro di tesi, si è eseguito un debug di Ne.Mo., un tool per l’analisi morfologica dei neuroni, che si è successivamente registrato presso la SIAE.

Si è saggiata la corretta della funzionalità di Ne.Mo. analizzando vetrini contenenti claustro, una sottile lamina di sostanza grigia situata nella corteccia dell’*insula*. I neuroni sono stati fotografati a 40X, per ottenere immagini il meno possibile affette da rumore di background, ma in questo modo non è stato possibile racchiudere nel campo visivo

della fotocamera l'intera cellula. Si sono quindi estratte più immagini che ritraevano varie regioni del neurone, e si è effettuato, con l'aiuto di un tool implementato in Matlab, un'operazione di registrazione delle stesse, ottenendo un'unica immagine idonea all'applicazione.

Si è quindi effettuato il pre-processing e l'analisi morfologica tramite Ne.Mo., raccogliendo i dati in un opportuno datamatrix, che è stato successivamente analizzato con la 3-way PCA.

Nella seconda fase del progetto di tesi, si è infatti implementato un plug-in che effettua tale tecnica. Questo era già stato implementato da Leardi e collaboratori in un editor Matlab: si è quindi realizzata un'operazione di traduzione da un linguaggio di programmazione all'altro, tenendo conto delle differenze sostanziali tra i due. La scelta del linguaggio di programmazione permette di caricare il plug-in in rete, condividendo con la comunità di Image J gli algoritmi implementati.

Per validare l'effettiva equivalenza del plug-in all'editor si sono analizzati vari datamatrix.

Si sono analizzate, infatti, quattro matrici di dati, tre delle quali contenevano informazioni sull'analisi morfologica dei neuroni. Nella quarta matrice, invece, sono stati organizzati i dati di esperimenti che saggiano la vitalità di tre tipi cellulari diversi in coltura, sottoposti a diverse concentrazioni di una particolare nanoparticella tossica, l'Ag NM300.

Il principale risultato è stato che analizzando i datamatrix sia con lo script in Matlab che con quello in Java, si sono ottenuti gli stessi identici risultati.

Inoltre l'osservazione dei grafici degli oggetti, delle variabili e delle condizioni ha permesso di trarre delle importanti considerazioni riguardanti i dati analizzati..

Per quanto riguarda i primi tre datamatrix, gli assi delle componenti principali sono stati interpretati assumendo che l'asse x rappresenti la direzionalità mentre l'asse y rappresenti la complessità dell'albero dendritico.

Sulla base di questa interpretazione, si è visto che le variabili che danno informazioni sulla direzionalità sono l'estensione, il percorso sull'asse e l'angolo del cono, mentre il numero massimo di intersezioni, il raggio critico e il coefficiente di regressione sono indici della complessità dell'albero dendritico.

Per quanto riguarda il grafico delle cellule, bisogna distinguere i tre datamatrix considerati: infatti, se nel primo si sono analizzate due colture di neuroni provenienti da topi wild-type, nel secondo le cellule di Purkinje erano estratte sia da topi wild-type che da modelli animali di autismo; nel terzo datamatrix, le cellule provenivano dal claustro, in cui è possibile individuare due diversi tipi di neuroni, uno che tende a svilupparsi con una maggiore direzionalità dell'albero dendritico rispetto all'altro.

Dal grafico delle cellule ottenuto applicando la 3-way PCA alla prima matrice, si è notato che per la maggior parte esse sono molto vicine tra loro, tuttavia ci sono alcune cellule che si distaccano dal gruppo, avendo esse una direzionalità maggiore o minore rispetto alla media. Si può ipotizzare che le cellule danneggiate o diverse dalla normale fisiologia neuronale abbiano una disposizione nel grafico diversa rispetto alla media, quindi possano essere identificate e classificate.

Ciò è più evidente dal grafico delle cellule ottenuto con il secondo datamatrix: le cellule estratte da topo Engrailed 2, infatti, risultano di dimensioni e struttura morfologica alterata ed appaiano in una posizione diversa del grafico rispetto alle cellule di topo wild-type.

Analizzando il grafico degli oggetti ottenuto applicando la 3-way PCA alla terza matrice, inoltre, è evidente che questa riesca effettivamente nella classificazione dei due tipi cellulari individuati nel claustro. Il metodo Tucker3, quindi, riesce nella classificazione di diversi tipi cellulari in base alla direzionalità, rilevando all'interno dei vari cluster ulteriori differenze, ad esempio dal punto di vista della complessità dendritica.

Infine, analizzando il grafico relativo alle condizioni, ovvero i giorni, delle prime due matrici, si è notato come durante la crescita cellulare aumentino complessità e direzionalità. Le cellule infatti crescendo sviluppano un maggior numero di ramificazioni, tendendo verso una

direzione preferenziale di sviluppo. Ciò è osservabile fino a un certo tempo t: dopo un certo numero di giorni dalla messa in coltura, infatti, i due parametri iniziano invece a diminuire, segno della sofferenza cellulare prima della morte, che porta a un danneggiamento dell'albero dendritico.

Per quanto riguarda l'ultimo datamatrix analizzato, si è anche in questo caso data un'interpretazione agli assi delle componenti principali: l'asse x è un indice della "resistenza" della cellula ad essere attaccata dalla nanoparticella, mentre l'asse y rappresenta la vitalità della cellula.

In base a questa interpretazione si sono analizzati i tre grafici.

Per quanto riguarda il plot delle condizioni, cioè i test per saggiare la vitalità cellulare a diverse concentrazioni di nanoparticella somministrata alle colture cellulari, è evidente come l'Alamar Blue e il test LDH diano informazioni complementari: mentre il primo, infatti, è un indice dell'attività metabolica, quindi è direttamente correlato alla vitalità cellulare, il secondo, essendo una misura della quantità di LDH nel medium di coltura che aumenta in caso di lisi cellulare, è inversamente correlato alla vitalità stessa.

Nel plot delle variabili, si evidenzia la netta presenza di due cluster, che raggruppano le concentrazioni di nanoparticella somministrate alle tre colture cellulari: la vitalità, in corrispondenza delle concentrazioni minori, è più alta rispetto che alle concentrazioni maggiori di Ag NM300.

Il risultato più significativo si ha, infine, guardando il grafico degli oggetti. Infatti, i tre tipi cellulari non esibiscono la stessa reazione alla somministrazione della nanoparticella nel medium di coltura. In particolare, gli epatociti sembrano essere meno "resistenti" alla presenza dell'Ag NM300 rispetto alle cellule endoteliali ed epiteliali intestinali. Ciò è confermato dalla letteratura a riguardo.

In definitiva, il plug-in per l'analisi 3-way PCA in Java è del tutto equivalente all'editor in Matlab che esegue l'analisi 3-way PCA.

Inoltre, tale tecnica è un potente strumento di classificazione ed analisi di dati multivariati.

Uno sviluppo futuro interessante di questo lavoro potrebbe riguardare l'applicazione della 3-way PCA a vari ambiti di studio, in particolare l'analisi della morfologia neuronale. Infatti, la comprensione del meccanismo di evoluzione di un neurone e l'estrazione delle variabili morfologiche che lo caratterizzano rappresentano il primo e più importante passo per l'individuazione dei fenomeni alla base dello sviluppo neurologico causata dalle patologie neurodegenerative o dai ritardi mentali. La classificazione che si ottiene utilizzando la 3-way PCA può, infatti, essere di aiuto alla comprensione delle alterazioni che si riscontrano in una particolare patologia, ma anche delle variabili più significative che le descrivono.
Un altro sviluppo futuro potrebbe riguardare il miglioramento delle tecniche software utilizzate per l'implementazione del plug-in, per renderlo computazionalmente più efficiente.
Inoltre, per quanto riguarda Ne.Mo., si potrebbe pensare di tradurre l'intero tool in linguaggio di programmazione Java. Il plug-in così implementato non vorrà sostituire gli algoritmi, per lo più già presenti in Image J, che portano all'estrazione dei dati morfologici a partire da immagini che ritraggono neuroni, sarà al contrario un tool ottimizzato per lo studio delle cellule neuronali, in fluorescenza o non, a partire dal pre-processing, fino alla visualizzazione dei risultati ottenuti tramite grafici. Tutto ciò mantenendo la principale caratteristica di Ne.Mo., cioè la semplicità nell'utilizzo anche per un utente non esperto di informatica.

Appendice A
Algoritmi per l'implementazione del plug-in

```
package PCA;
import ij.gui.GenericDialog;

import ij.IJ;
import ij.ImagePlus;

import java.awt.AWTEvent;
import java.io.File;
import java.io.IOException;
import java.math.BigDecimal;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Map;
import java.util.Set;

import jmathlib.toolbox.jmathlib.matrix._private.Jampack.*;

import ij.gui.Plot;
import ij.io.FileInfo;
import ij.io.OpenDialog;
import ij.io.Opener;
import ij.process.ImageProcessor;

import java.awt.Color;

import org.netlib.util.intW;

import com.jmatio.io.MatFileReader;
import com.jmatio.types.MLArray;
import com.jmatio.types.MLDouble;

public class MainPCA {

	private static boolean jScaling = false;
	private static int nConditions = 1;
	private static int nIterations = 20;

	public static void main(String[] arg) throws JampackException {

		OpenDialog dialog = new OpenDialog("Select a file", "");
		String dataFilename = dialog.getFileName();
		if( dataFilename == "")
		{
			IJ.showMessage("No file is selected!");
```

```
                return;
            }
            double [][] x = getImportData(new File(dataFilename));
            showDialog();

            double [][] xo;
            double [][] xv;
            double [][] xc;
            double [][] app;
            int nRows = x.length;    //numero righe
            int nCols = x[0].length;    //numero colonne

            if (jScaling == true) {   //x = autosc(x)

                double [] average = new double [nCols];
                double [] stdev = new double [nCols];

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

                        average[i] += x[j][i];
                    }
                    average[i] = average[i]/nRows;
                }

                for (int i = 0; i<nCols; i++){
                    for (int j = 0; j<nRows; j++){
                        stdev[i] += (x[j][i]-
average[i])*(x[j][i]-average[i]);
                    }
                    stdev[i] = Math.sqrt(stdev[i]/(nRows-1));
                }

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

                        x[i][j] = (x[i][j] - average[j]) /
stdev[j];
                    }
                }
            }


            int o = nRows/nConditions;

            xo = new double [o][nConditions*nCols];
            xv = new double [nCols][nConditions*o];

            for(int p=0; p<nConditions; p++) {
                for(int i=0; i<nCols; i++) {
```

```
                        for (int j=0; j<o; j++) {

                                xo[j][i+p*nCols] = x[j+p*o][i];

                                xv[i][j+p*o] = xo[j][i+p*nCols];
                    }
                }
            }

            double [] t = new double [o*nCols];
            xc = new double [nConditions][o*nCols];

            for (int r = 0; r<nConditions; r++) {
                for (int p = 0; p<o; p++) {
                    for (int i = 0; i<nCols; i++) {

                        t[i+p*nCols] = xv[i][r*o+p];

                    }
                }

                for(int i = 0; i<nCols*o; i++){

                    xc [r][i] = t[i];
                }
            }

            double varin = 0;

            for (int i = 0; i<nRows; i++) {
                for (int j = 0; j<nCols; j++) {
                    varin += x[i][j]*x[i][j];
                };
            };

            GenericDialog gdiag1 = new GenericDialog ("Initial
variance");
            gdiag1.addMessage("Initial variance is:  " +round(varin,
3));
            gdiag1.showDialog();


            double [][] lo = new double [2][xo.length];
            lo = getAutovector(getMatrixTraspose(xo));

            double [][] lv = new double [2][xv.length];
        lv = getAutovector(getMatrixTraspose(xv));

    double [][] lc = new double [2][xc.length];
    lc = getAutovector(getMatrixTraspose(xc));
```

```
	double [][] g = new double [lo.length][lc.length*lv.length];
		double [][] g2 = new double [2][4];
		double d = 10000;
		int ci = 0;

		LinkedList<Double> perc = new LinkedList<Double>();
		double sscore = 0;

		double [][] gtrasp= new double [g[0].length][g.length];
		int bmax = 0;
		double amax = 0;
		int f = 0;

		while( round(d, 3) > 0.001 & ci < (nIterations+1) )
//round(d, 3) approssima d a 3 cifre decimali
		{
			if (ci > 0)
			{
				double [][] m = new double
[xo.length][xo.length]; //m = xo*kron*xo'
				m =
getMatrixProduct(getMatrixProduct(xo,
kron(getMatrixProduct(getMatrixTraspose(lc), lc),
getMatrixProduct(getMatrixTraspose(lv), lv))),
getMatrixTraspose(xo));

				app = new double [lo.length][lo.length];
				app = getMatrixProduct(getMatrixProduct
(lo, getMatrixProduct(m, m)), getMatrixTraspose(lo));


				Zmat zInv = Inv.o(sqrtMatrix(app));
				app = zInv.getRe();

				lo =
getMatrixTraspose(getMatrixProduct(getMatrixProduct(m,
getMatrixTraspose(lo)), app));

				m = new double [xv.length][xv.length];
				m =
getMatrixProduct(getMatrixProduct(xv,
kron(getMatrixProduct(getMatrixTraspose(lc), lc),
getMatrixProduct(getMatrixTraspose(lo), lo))),
getMatrixTraspose(xv));

				app = new double [lv.length][lv.length];
//lv*m^2*lv'
				app =
getMatrixProduct(getMatrixProduct(lv, getMatrixProduct(m, m)),
getMatrixTraspose(lv));
```

```
                    Zmat zInv1 = Inv.o(sqrtMatrix(app));
                    app = zInv1.getRe();

                    lv = getMatrixTraspose(getMatrixProduct
(getMatrixProduct(m, getMatrixTraspose(lv)), app));

                    m = new double [xc.length][xc.length];
                    m =
getMatrixProduct(getMatrixProduct(xc,
kron(getMatrixProduct(getMatrixTraspose(lo), lo),
getMatrixProduct(getMatrixTraspose(lv), lv))),
getMatrixTraspose(xc));

                    app = new double [lc.length][lc.length];
                    app =
getMatrixProduct(getMatrixProduct(lc, getMatrixProduct(m, m)),
getMatrixTraspose(lc));

                    Zmat zInv2 = Inv.o(sqrtMatrix(app));
                    app = zInv2.getRe();

                    lc =
getMatrixTraspose(getMatrixProduct(getMatrixProduct(m,
getMatrixTraspose(lc)), app));

                }//if(ci>0)

                g = getMatrixProduct(getMatrixProduct(lo, xo),
kron(getMatrixTraspose(lc), getMatrixTraspose(lv)));
                gtrasp = getMatrixTraspose (g);


                for (int i = 0; i<gtrasp.length; i++){
                    for (int j = 0; j<gtrasp[0].length; j++){
                        gtrasp[i][j] = Math.abs(gtrasp[i][j]);
//minimo valore 0
                    }
                }

                amax = gtrasp[0][0];    //ehm...
                for (int i = 0; i<gtrasp.length; i++){
                    for (int j = 0; j<gtrasp[0].length; j++){
                        if (gtrasp[i][j]> amax){
                            f = j;
                            amax = gtrasp [i][j];
                            bmax = f;
                        }
                    }
                }
```

```
                    if (bmax == 1)
                    {

                            double [][] lo2 = new double
[lo.length][lo[0].length];

                            for (int i = 0; i<lo2[0].length; i++){
                                    lo2[0][i] = lo[1][i];
                                    lo2[1][i] = lo[0][i];
                            }

                            lo = (double[][])lo2.clone();

                            g = getMatrixProduct(getMatrixProduct(lo,
xo), kron(getMatrixTraspose(lc), getMatrixTraspose(lv)));
                    } // if(bmax == 1)

                    for (int i = 0; i<g.length; i++){
                            for (int j = 0; j<g[0].length; j++){
                                    g[i][j] = Math.abs(g[i][j]);
//minimo valore 0
                            }
                    }

                    amax = g[0][0];     //ehm...
                    for (int i = 0; i<g.length; i++){
                            for (int j = 0; j<g[0].length; j++){
                                    if (g[i][j]> amax){
                                            f = j;
                                            amax = g [i][j];
                                            bmax = f;
                                    }
                            }
                    }

                    if (bmax == 1 | bmax == 3)
                    {
                            double [][] lc2 = new double
[lc.length][lc[0].length];

                            for (int i = 0; i<lc2[0].length; i++){
                                    lc2[0][i] = lc[1][i];
                                    lc2[1][i] = lc[0][i];
                            }

                            lc = (double[][])lc2.clone();

                            g = getMatrixProduct(getMatrixProduct(lo,
xo), kron(getMatrixTraspose(lc), getMatrixTraspose(lv)));

                    } //if(b == 1 | b == 3)
```

```
                    if (bmax>1){

                            double [][] lv2 = new double
[lv.length][lv[0].length];
                            for (int i = 0; i<lv[0].length; i++){
                                    lv2[0][i] = lv[1][i];
                                    lv2[1][i] = lv[0][i];
                            }

                            lv = (double[][])lv2.clone();

                    } //if(bmax>1)

                    //questa istruzione è corretta metterla fuori dall'if
                    //nei cicli precedenti era sempre all'interno ----->
vedi codice matlab
                    g = getMatrixProduct(getMatrixProduct(lo, xo),
kron(getMatrixTraspose(lc), getMatrixTraspose(lv)));

                    sscore = 0;
                    for (int i = 0; i<g.length; i++) {
                            for (int j = 0; j<g[0].length; j++){
                                    sscore += g[i][j]*g[i][j];
                            };
                    };

                    double percvarexp = (sscore/varin)*100;

                    GenericDialog gdiag = new GenericDialog
("Output Dialog Box");
                    gdiag.addMessage("%expl. var. is  " +
round(percvarexp, 3));
                    gdiag.showDialog();

                    perc.add(ci, percvarexp);

                    if (ci>0){
                            d = round(perc.get(ci) - perc.get(ci-1), 3);
                    };

                    ci = ci+1;
            }; //while(d>0.001 & ci<maxci+1)

            double [] vx = new double [perc.size()];
            double [] percArray = new double [perc.size()];
            for (int i = 0; i<perc.size(); i++){
                    vx[i] = i;
                    percArray[i] = perc.get(i);
            };
```

```
		Plot plot4 = new Plot ("Evolution of the explained
variance", "", "", vx, percArray);
		plot4.setLimits(getMin(vx), getMax(vx),
getMin(percArray), getMax(percArray));
		plot4.setColor(Color.blue);
		plot4.show();

		double fi = (g[0][0]*g[0][0]+ g[1][3]*g[1][3])/sscore;

		double fin = 0;
		double fimax = fi;

		while (round(fimax, 3)>round(fin, 3)){
			fin = fimax;

			double rotb = 0;
			double [][] lob = new double
[lo.length][lo[0].length];

			for (int rot = -90; rot<90; rot++){

				double [][] rm = new double [2][2];
				rm[0][0] = Math.cos((rot*Math.PI)/180);
				rm[0][1] = - (Math.sin((rot*Math.PI)/180));
				rm[1][0] = Math.sin((rot*Math.PI)/180);
				rm[1][1] = Math.cos ((rot*Math.PI)/180);

				double [][] lo2 = new
double[rm.length][lo[0].length];
				lo2 = getMatrixProduct(rm, lo);

				g2 = new double
[lo2.length][lc.length*lv.length];
				g2 =
getMatrixProduct(getMatrixProduct(lo2, xo),
kron(getMatrixTraspose(lc), getMatrixTraspose(lv)));

				double g2square = 0;
				for (int i = 0; i<g2.length; i++) {

					for (int j = 0; j<g2[0].length; j++){
						g2square +=
(g2[i][j]*g2[i][j]);
					}
				}

				double fi2 = (g2[0][0]*g2[0][0]+
g2[1][3]*g2[1][3])/g2square;

				if (round(fi2, 3)>round(fimax, 3)){
```

```
                                lob = (double[][])lo2.clone();
                                fimax = fi2;
                                rotb = rot;
                        }
                }

                if (rotb != 0){
                        lo = (double[][])lob.clone();
                }

                //ROTATION OF VARIABLES
                rotb = 0;
                double [][] lvb = new double
[lv.length][lv[0].length];

                for (int rot = -90; rot<90; rot++){

                        double [][] rm = new double [2][2];
                        rm[0][0] = Math.cos((rot*Math.PI)/180);
                        rm[0][1] = -(Math.sin((rot*Math.PI)/180));
                        rm[1][0] = Math.sin((rot*Math.PI)/180);
                        rm[1][1] = Math.cos((rot*Math.PI)/180);

                        double [][] lv2 = new
double[rm.length][lv[0].length];
                        lv2 = getMatrixProduct(rm, lv);

                        g2 = new double
[lo.length][lc.length*lv2.length];
                        g2 =
getMatrixProduct(getMatrixProduct(lo, xo),
kron(getMatrixTraspose(lc), getMatrixTraspose(lv2)));

                        double g2square = 0;
                        for (int i = 0; i<g2.length; i++) {

                                for (int j = 0; j<g2[0].length; j++){
                                        g2square +=
(g2[i][j]*g2[i][j]);
                                }
                        }

                        double fi2 =
(g2[0][0]*g2[0][0]+g2[1][3]*g2[1][3])/g2square;

                        if (round(fi2, 3)>round(fimax, 3)){

                                lvb = (double[][])lv2.clone();
                                fimax = fi2;
                                rotb = rot;
                        }
```

```
                    }

                    if (rotb != 0){
                        lv = (double[][])lvb.clone();
                    }

                    //ROTATION OF CONDITIONS
                    rotb = 0;
                    double [][] lcb = new double
[lc.length][lc[0].length];

                    for (int rot = -90; rot<90; rot++ ){
                        double [][] rm = new double [2][2];
                        rm[0][0] = Math.cos((rot*Math.PI)/180);
                        rm[0][1] = -(Math.sin((rot*Math.PI)/180));
                        rm[1][0] = Math.sin((rot*Math.PI)/180);
                        rm[1][1] = Math.cos((rot*Math.PI)/180);

                        double[][] lc2 = new double
[rm.length][lc[0].length];
                        lc2 = getMatrixProduct(rm, lc);

                        g2 = new double
[lo.length][lc2.length*lv.length];
                        g2 =
getMatrixProduct(getMatrixProduct(lo, xo),
kron(getMatrixTraspose(lc2), getMatrixTraspose(lv)));

                        double g2square = 0;
                        for (int i = 0; i<g2.length; i++) {

                            for (int j = 0; j<g2[0].length; j++){

                                g2square +=
(g2[i][j]*g2[i][j]);
                            }
                        }

                        double fi2 =
(g2[0][0]*g2[0][0]+g2[1][3]*g2[1][3])/g2square;

                        if (round(fi2, 3)>round(fimax, 3)){

                            lcb = (double[][])lc2.clone();

                            fimax = fi2;
                            rotb = rot;
                        }
                    }

                    if (rotb != 0){
```

```
                lc = (double[][])lcb.clone();
        }


} //while (fimax>fin)


int nc = lo.length;
double l = 0;
double t1 = 0;

for (int z = 0; z<nc; z++){

        double [] ap = new double [lo[0].length];

        for (int i = 0; i<lo[0].length; i++){

                ap[i] = lo[z][i];
        }

        double emme = getMean(ap);
        for (int i = 0; i<nCols; i++){
                l = 0;
                for(int w = 0; w<o; w++){
                        int dim = ((nRows-w-1)/2)+1;
                        ap = new double [dim];
                        int su = 0;

                        for (int j=w; j<nRows; j=j+o){

                                        ap[su] = x[j][i];
                                        su += 1;
                        }

                        l += getMean(ap)*(lo[z][w]-emme);
                }
                t1 += l*lv[z][i];
        }

        if (round(t1, 3)<0){
                for(int i = 0; i<lo[0].length;i++){

                        lo[z][i] = -lo[z][i];
                }
        }

        emme = getMean(lc[z]);
        t1 = 0;

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

```
                        for (int w = 0; w<nConditions; w++){

                                int dim = o;
                                ap = new double [dim];
                                int su = 0;

                                for (int j = w*o; j<(w+1)*o; j++){
                                        ap[su] = x[j][i];
                                        su += 1;
                                }

                                l += getMean(ap)*(lc[z][w]-emme);
                        }

                        t1 += l*lv[z][i];
                }

                if (round(t1, 3)<0){
                        for(int i = 0; i<lc[0].length; i++){
                                lc[z][i] = -lc[z][i];
                        }
                }
        }

        g = getMatrixProduct(getMatrixProduct(lo, xo),
kron(getMatrixTraspose(lc), getMatrixTraspose(lv)));
        gtrasp = getMatrixTraspose(g);

        for (int i = 0; i<gtrasp.length; i++){
                for (int j = 0; j<gtrasp[0].length; j++){
                        gtrasp[i][j] = Math.abs(gtrasp[i][j]);
                }
        }

        amax = gtrasp[0][0];
        for (int i = 0; i<gtrasp.length; i++){
                for (int j = 0; j<gtrasp[0].length; j++){
                        if (gtrasp[i][j]> amax){
                                f = j;
                                amax = gtrasp [i][j];
                                bmax = f;
                        }
                }
        }

        if (bmax == 1){

                double[][] lo2 = new
double[lo.length][lo[0].length];

                for (int i = 0; i<lo[0].length; i++){
```

```
                    lo2[0][i] = lo[1][i];
                    lo2[1][i] = lo[0][i];
                }


                lo = (double[][])lo2.clone();

                g = getMatrixProduct(getMatrixProduct(lo, xo),
kron(getMatrixTraspose(lc), getMatrixTraspose(lv)));

            }

                double [][] gap = new double
[g.length][g[0].length];
                for (int i = 0; i<g.length; i++){

                    for (int j = 0; j<g[0].length; j++){

                        gap[i][j] = Math.abs(g[i][j]);
                    }
                }

                amax = gap[0][0];
                for (int i = 0; i<gap.length; i++){

                    for (int j = 0; j<gap[0].length; j++){

                        if (gap[i][j]> amax){

                            f = j;
                            amax = gap[i][j];
                            bmax = f;
                        }
                    }
                }

                if (bmax == 1 | bmax == 3)
                {
                    double[][] lc2 = new
double[lc.length][lc[0].length];

                    for (int i = 0; i<lc[0].length; i++){

                        lc2[0][i] = lc[1][i];
                        lc2[1][i] = lc[0][i];
                    };

                    lc = (double[][])lc2.clone();

                }
```

```
                    if (bmax>1)
                    {
                            double[][] lv2 = new double
[lv.length][lv[0].length];

                            for (int i = 0; i<lv[0].length; i++){

                                    lv2[0][i] = lv[1][i];
                                    lv2[1][i] = lv[0][i];
                            }

                      lv = (double[][])lv2.clone();


                    }


    /*****************************************************
**/

                    int ka = 1;
                    String tex = "Plot of objects";
                    plotTwoComponents(getMatrixTraspose(lo), ka,
tex);

                    ka = 2;
                    tex = "Plot of variables";
                    plotTwoComponents(getMatrixTraspose(lv), ka,
tex);

                    ka = 3;
                    tex = "Plot of Conditions";
                    plotTwoComponents(getMatrixTraspose(lc), ka,
tex);

//                  g = getMatrixProduct(getMatrixProduct(lo, xo),
kron(getMatrixTraspose(lc), getMatrixTraspose(lv)));
//
//                  //immagine di background.
//                  Opener opener = new Opener ();
//                  ImagePlus imp =
opener.openImage("C:\\Documents and
Settings\\chiara\\Documenti\\workspace\\3way_PCA\\prova.bmp");
//                  imp.show();
//                  FileInfo fInfo = imp.getFileInfo();

//                  System.out.println("Height: "+fInfo.height);
//                  System.out.println("Pixel height:
"+fInfo.pixelHeight);
```

```
//                      System.out.println("fInfo.fileFormat: " +
fInfo.fileFormat);
//                      System.out.println("fInfo.samplesPerPixel:
"+fInfo.samplesPerPixel);
//                      ImageProcessor ip = imp.getChannelProcessor();
//                      Color value = new Color (0, 255, 255);
//                      ip.setColor(value);
//                      ip.drawLine(10, 10, 20, 20);
//                      ip.drawDot(10, 10);
//                      ip.drawOval(30, 30, 20, 20);


                        double [][] lotrasp = getMatrixTraspose(lo);

                        double [][] lvtrasp = getMatrixTraspose(lv);

                        double [][] lctrasp = getMatrixTraspose(lc);

                        double [][] xt = new double
[lotrasp.length+lvtrasp.length+lctrasp.length][lotrasp[0].length];

                        for (int i = 0; i<lotrasp[0].length; i++){

                                for (int j = 0; j<lotrasp.length; j++){
                                        xt[j][i] = lotrasp[j][i];
                                };

                                for (int j = 0; j<lvtrasp.length; j++){
                                        xt[lotrasp.length+j][i] = lvtrasp[j][i];
                                };

                                for (int j = 0; j<lctrasp.length; j++){
                                        xt[lotrasp.length+lvtrasp.length+j][i]
= lctrasp[j][i];
                                }
                        }

                        double [][] rm = new double
[o*nConditions][nCols];
                        double rv = 0;

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

                                        for (int kappa = 0;
kappa<nConditions; kappa++){
                                                rv =
lo[0][i]*lv[0][j]*lc[0][kappa]*g[0][0];
                                                rv +=
lo[0][i]*lv[1][j]*lc[0][kappa]*g[0][1];
```

```
                                        rv +=
lo[0][i]*lv[0][j]*lc[1][kappa]*g[0][2];
                                        rv +=
lo[0][i]*lv[1][j]*lc[1][kappa]*g[0][3];
                                        rv +=
lo[1][i]*lv[0][j]*lc[0][kappa]*g[1][0];
                                        rv +=
lo[1][i]*lv[1][j]*lc[0][kappa]*g[1][1];
                                        rv +=
lo[1][i]*lv[0][j]*lc[1][kappa]*g[1][2];
                                        rv +=
lo[1][i]*lv[1][j]*lc[1][kappa]*g[1][3];
                                        rm[o*kappa+i][j] = rv;

                                }
                        }
                }

                double [][] di = new double
[rm.length][rm[0].length];
                for (int i = 0; i<rm.length; i++){

                        for (int j = 0; j<rm[0].length; j++){

                                di[i][j] = (rm[i][j]-x[i][j])*(rm[i][j]-
x[i][j]);
                        }
                }

                double [] rvi = new double [nCols];

                double [][] ro = new double [o][o];

                double [][] rc = new double
[nConditions][nConditions];

                for (int i = 0; i<di[0].length; i++){

                        for(int j = 0; j<di.length; j++){

                                rvi[i] += di[j][i];
                        }
                }

                for (int i = 0; i<rvi.length; i++){

                        rvi[i] = rvi[i]/(o*nConditions);
                }

                double [] roc = new double [di.length];
```

```
                    for (int j = 0; j<di.length; j++){

                        for(int i = 0; i<di[0].length; i++){

                            roc[j] += di[j][i];
                        }
                    }

                    for (int i = 0; i<o; i++){
                        for(int j = i; j<o*nConditions; j=j+o){

                            ro[i][0] += roc[j];
                        }
                    }

                    for (int i = 0; i<o; i++){
                        for(int j = 0; j<o; j++){
                            ro[i][j] =
ro[i][j]/(nConditions*nCols);
                        }
                    }

                    for (int i = 0; i<nConditions; i++){

                        for(int j = i*o; j<o*(i+1); j++){

                            rc[i][0] += roc[j];
                        }
                    }

                    for (int i = 0; i<nConditions; i++){

                        for(int j = 0; j<nConditions; j++){

                            rc[i][j] = rc[i][j]/(o*nCols);
                        }
                    }


                    double [] obj;
                    obj = new double [ro.length];
                    for(int i = 0; i<ro.length; i++){

                        obj[i] = i+1;
                    }

                    Plot plot6 = new Plot("RMSE of objects", "Axis1",
"Axis2", new double[]{0.0}, new double[]{0.0});
                    plot6.setLimits(getMin(obj)-1, getMax(obj)+1, 0,
getMax(getMatrixTraspose(ro)[0])+1);
```

```
                    plot6.addPoints(obj, getMatrixTraspose(ro)[0],
Plot.CIRCLE);

                    for (int i = 0; i<ro[0].length; i++){
                            plot6.drawLine(i+1, ro[i][0], i+1, 0);
                    };


                    plot6.show();

                    double [] var;
                    var = new double [rvi.length];
                    for(int i = 0; i<rvi.length; i++){

                            var[i] = i+1;
                    };

                    Plot plot7 = new Plot("RMSE of variables",
"Axis1", "Axis2", new double[]{0.0}, new double[]{0.0});
                    plot7.setLimits(getMin(var)-1, getMax(var)+1, 0,
getMax(rvi)+1);
                    plot7.addPoints(var, rvi, Plot.CIRCLE);
                    plot7.addPoints(var, rvi, Plot.CIRCLE);

                    for (int i = 0; i<rvi.length; i++){
                            plot7.drawLine(i+1, rvi[i], i+1, 0);
                    };

                    plot7.show();


                    double [] cond;
                    cond = new double [rc.length];
                    for(int i = 0; i<rc.length; i++){

                            cond[i] = i+1;
                    };


                    Plot plot8 = new Plot("RMSE of conditions",
"Axis1", "Axis2", new double[]{0.0}, new double[]{0.0});
                    plot8.setLimits(getMin(obj)-1, getMax(obj)+1, 0,
getMax(rc[0])+1);
                    plot8.addPoints(cond, getMatrixTraspose(rc)[0],
Plot.CIRCLE);

                    for (int i = 0; i<rc[0].length; i++){
                            plot8.drawLine(i+1, rc[i][0], i+1, 0);
                    };
```

```
                plot8.show();


        }


    /* MATLAB data matrix */
    public static double [][] getImportData(File f) {

        double[][] data = null;

        if(f.getName().endsWith(".mat"))
        {
            MatFileReader matReader = null;
            Map<String, MLArray> matrix = null;
            try {
                matReader = new MatFileReader(f);
                matrix = matReader.read(f);

            } catch (IOException e) {
                IJ.log(""+e);
                return null;
            }

            Set<String> s = matrix.keySet();
            int selectedIndex = 0;
            if(s.size() > 1)
            {
                //visualizza dialog per selezionare la
matrice
                //(Ho visto che si possono salvare più
matrici in un file .mat.
                //Se è necessario gestiamo questo caso
altrimenti diamo per
                //scontato che ci sia una matrice sola per
ogni file .mat)

                //selcetedIndex = ...
            }

            data =
((MLDouble)matReader.getMLArray((String)
s.toArray()[selectedIndex])).getArray();
        }
        else if(f.getName().endsWith(".txt"))
        {
            data = new double[1][1];
        }

        return data;
```

```
    }
    /* */


    /* Dialog parameters */
    public static void showDialog() {
     GenericDialog gd = new GenericDialog("3-Way PCA");
     gd.addCheckbox("Scaling", jScaling);
     gd.addNumericField("# Conditions (Submultiple of row
number):", nConditions, 0);
     gd.addNumericField("# Iterations:", nIterations, 0);
     gd.showDialog();
     jScaling = gd.getNextBoolean();
     nConditions = (int)gd.getNextNumber();
     nIterations = (int)gd.getNextNumber();
   };

    public static boolean dialogItemChanged(GenericDialog gd,
AWTEvent e) {
            if (IJ.isMacOSX()) IJ.wait(100);
            jScaling = gd.getNextBoolean();
            nConditions = (int)gd.getNextNumber();
            nIterations = (int)gd.getNextNumber();
            if (gd.invalidNumber())
                    return false;
            else
                    return true;
    };
    /* */


    /* Math utils */
    public static double round(double d, int decimalPlace) {
       BigDecimal bd = new BigDecimal(d);
       bd = bd.setScale(decimalPlace,
BigDecimal.ROUND_HALF_UP);
       return bd.doubleValue();
    }

    public static double getMean(double [] array) {
            double mean = 0;
            for (int i = 0 ; i<array.length; i++){
                    mean += array[i];
            };
            return mean/(array.length);
    }

    public static double getMax(double [] array) {
            double max = array[0];
            for (int i = 1 ; i<array.length; i++){
                    if(array[i] > max)
```

```
                        max = array[i];
                }
                return max;
        }

        public static double getMin(double [] array) {
                double min = array[0];
                for (int i = 1 ; i<array.length; i++){
                        if(array[i] < min)
                                min = array[i];
                }
                return min;
        }

        public static double [][] getMatrixProduct(double [][] A, double
[][] B) {
                double[][] product = new double [A.length][B[0].length];
                for (int i = 0; i<product.length; i++){
                        for (int j = 0; j<product[0].length; j++){
                                product [i][j] = 0;
                                for (int k = 0; k<A[0].length; k++){
                                        product[i][j] += A[i][k]*B[k][j];
                                };
                        };
                }
                return product;
        };

        public static double [][] getMatrixTraspose(double [][] m) {
                double[][] mTrasp = new double [m[0].length][m.length];
                for (int i = 0; i<mTrasp.length; i++){
                        for (int j = 0; j<mTrasp[0].length; j++){
                                mTrasp[i][j] = m[j][i];
                        };
                };
                return mTrasp;
        };

        //kron(X,Y) returns the Kronecker tensor product of X and Y.
        //The result is a large array formed by taking all possible products
        //between the elements of X and those of Y.
        //If X is m-by-n and Y is p-by-q, then kron(X,Y) is m*p-by-n*q.
        public static double[][] kron(double[][] m1, double [][] m2){
                double [][] krone = new double
[m1.length*m2.length][m1[0].length*m2[0].length];
                for (int i = 0; i<m1.length; i++){
                        for (int j = 0; j<m1[0].length; j++){

                                for (int k = 0; k<m2.length; k++){
                                        for (int z = 0; z<m2[0].length; z++){
```

```
        krone[k+i*(m2.length)][z+j*(m2[0].length)] = m1[i][j]*m2[k][z];
                                        };
                                };
                        };
                };
           return krone;
        };

        public static double [][] getAutovector (double [][] xt){

                int r = xt.length; //numero di righe
                int c = xt[0].length;  //numero di colonne - il numero di
colonne è lo stesso per costruzione
                int t = 0;  //t inizializzato a 0
                double vartot = 0;


                for (int i = 0; i<r; i++) {
                        for (int j = 0; j<c; j++){
                                vartot += xt[i][j]*xt[i][j];
                        };
                };

                double [][] smat = new double [r][2];
           double [][] lmat = new double [2][c];

           double[] xmax = new double [r];  //riga    //xmax = xt(:, b(c))
           double [] rmax = new double [c];   //vettore riga
           double [] ss = new double [c];
           double [] a = new double [c];
           int[] b = new int [c];
                double [] vp = new double [2];  //while t<2..

                while (t<2) {
                        t = t+1;

                        Arrays.fill(ss, 0);

                        for (int i = 0; i<c; i++){    //sum(xt.^2)

                                for (int j = 0; j<r; j++){

                                        ss[i] += xt[j][i]*xt[j][i];
                                }
                        }

                        a = ss.clone();

                        Arrays.fill(b, 0);
```

```
                Arrays.sort(a);

                for (int i = 0; i<c; i++){  //ss
                    for(int j = 0; j<c; j++){                    //a
                            if (ss[i] == a[j]){

                                    b[j] = i;
                            }
                    }
                }

                Arrays.fill(xmax, 0);

                for (int i = 0; i<r; i++){
                    xmax[i] = xt[i][b[c-1]];
                }

                double s = 0;

                for (int i = 0; i<xmax.length; i++){    //s = xmax' * xmax

                    s += xmax[i]*xmax[i];
                }

                double diff = 1000;
                Arrays.fill(rmax, 0);

                while (diff > 0.0000001){

                    double [][] appoggio = new double [r][c];  //matrice
xt/s

                    for (int i = 0; i<r; i++){     // xt/s - matrice

                            for (int j = 0; j<c; j++){

                                    appoggio [i][j] = xt[i][j]/s;

                            }
                    }

                    double [] ap1 = new double [c];

                    for (int i = 0; i<c; i++){    //rmax = xmax'
*(xt/s=appoggio)
                            for (int k = 0; k<r; k++){

                                    ap1[i] += xmax[k] * appoggio[k][i];
                            }
                            rmax[i] = ap1[i];
                    }
```

```
                double rmaxsq = 0;   //rmaxsq = rmax*rmax'

                   for (int i = 0; i<c; i++) {
                         rmaxsq += rmax[i]*rmax[i];
                   }

                   for (int i = 0; i<c; i++) {
                         rmax[i] = rmax[i] / Math.sqrt(rmaxsq);
                   }

                   double app;


                   for (int i = 0; i<r; i++){    //xmax = xt*rmax'
                         app = 0;
                         for (int j = 0; j<c; j++){

                               app += xt[i][j]*rmax[j];
                         }
                         xmax[i] = app;
                }

                double s2 = 0;


                         for (int i = 0; i<xmax.length; i++) {    //s =
xmax' * xmax
                               s2 += xmax[i]*xmax[i];
                         }

                         diff = Math.abs(s2-s);  //diff = abs(s2-s)
                   s = s2;
             }  //while interno

             for (int i = 0; i<r; i++){
                   smat[i][t-1] = xmax[i];
             }

             for (int i = 0; i<c; i++){
                   lmat[t-1][i] = rmax[i];
             }


             double [] varexp = new double [2];
             varexp[t-1] = 0;

             for (int i = 0; i<r; i++){    //s = xmax' * xmax

                   varexp[t-1] += (xmax[i] * xmax[i]);
             }
```

```
            vp[t-1] = (varexp[t-1] / vartot) * 100;

               for(int i =0; i<r; i++){
                     for(int j = 0; j<rmax.length; j++){

                              xt[i][j] = xt[i][j]-xmax[i]*rmax[j];
                     }
               }
            };   //while esterno

            for (int i = 0; i<2; i++){
                     System.out.println("vp = " + vp[i] + " "  + t); //disp
(vp)
            }

            double [] cumsum_vp = new double [t];
            cumsum_vp[0] = vp[0];
            cumsum_vp[1] = vp[0]+vp[1];


            for (int i = 0; i<cumsum_vp.length; i++){
                     System.out.println("cumsum vp = " +
cumsum_vp[i] + " " + t);
            }


            double b1 = smat[0].length;  //numero colonne

            if (b1 >1) {
                     double [] sma1 = new double [smat.length];
                     double [] sma2 = new double [smat.length];

                     for (int i = 0; i<sma1.length; i++){
                              sma1[i] = smat[i][0];
                              sma2[i] = smat[i][1];
                     }   //to plot


    /**************************************************
*****/

                     double [] totcum = new double [2];
                     totcum [0] = vp[0];
                     totcum [1] = vp[0] + vp[1];


    /**************************************************
*****/
```

```
                    Plot plot = new Plot ("Object scores on
eigenvectors 1-2."+ totcum[1] +"% of total variance", "sma1",
"sma2", new double[]{0.0}, new double[]{0.0});
                    plot.setLimits(getMin(sma1)-0.5,
getMax(sma1)+0.5, getMin(sma2)-0.5, getMax(sma2)+0.5);
                    plot.setColor(Color.red);
                    plot.addPoints(sma1, sma2, Plot.CIRCLE);

                    //if (getMin(sma1)<0 & getMax(sma2)>0 &
getMin(sma2)<0 & getMax(sma2)>0){

    //plot.addLabel(Math.abs(getMin(sma1)+0.5)/(Math.abs(getMin(s
ma1))+Math.abs(getMax(sma1))+1),
Math.abs(getMax(sma2)+0.5)/(Math.abs(getMin(sma2))+Math.abs(ge
tMax(sma2))+1), "+");
                    //}

                    if (getMin(sma1)<0 & getMax(sma1)>0 &
getMin(sma2)<0 & getMax(sma2)>0 ){
                        double cost1;
                        for (int i = 0; i<sma1.length; i++){
                            cost1 = 0;
                            for (int j = 0; j<i; j++){
                                if (sma1[i] == sma1[j] &
sma2[i] == sma2[j]){
                                    cost1 += 0.07;
                                };
                            };

    plot.addLabel((Math.abs(getMin(sma1))+sma1[i]+0.5+cost1)/(M
ath.abs(getMin(sma1))+Math.abs(getMax(sma1))+1),
(Math.abs(getMax(sma2))-
sma2[i]+0.5)/(Math.abs(getMin(sma2))+Math.abs(getMax(sma2))+1),
"" + i);
                        };
                    };

                    if (getMin(sma1)<0 & getMax(sma1)>0 &
(getMin(sma2)<0 & getMax(sma2)<0 || getMin(sma2)>0 &
getMax(sma2)>0 )){
                        double cost1;
                        for (int i = 0; i<sma1.length; i++){
                            cost1 = 0;
                            for (int j = 0; j<i; j++){
                                if (sma1[i] == sma1[j] &
sma2[i] == sma2[j]){
                                    cost1 += 0.07;
                                };
                            };

    plot.addLabel((Math.abs(getMin(sma1))+sma1[i]+0.5+cost1)/(M
```

```
ath.abs(getMin(sma1))+Math.abs(getMax(sma1))+1),
(Math.abs(getMax(sma2))-sma2[i]+0.5)/(-
Math.abs(getMin(sma2))+Math.abs(getMax(sma2))+1), "" + i);
                        };
                };

                if ((getMin(sma1)<0 & getMax(sma1)<0 ||
getMin(sma1)>0 & getMax(sma1)>0) & getMin(sma2)<0 &
getMax(sma2)>0 ){
                        double cost1;
                        for (int i = 0; i<sma1.length; i++){
                                cost1 = 0;
                                for (int j = 0; j<i; j++){
                                        if (sma1[i] == sma1[j] &
sma2[i] == sma2[j]){
                                                cost1 += 0.07;
                                        };
                                };

    plot.addLabel((Math.abs(getMin(sma1))+sma1[i]+0.5+cost1)/(-
Math.abs(getMin(sma1))+Math.abs(getMax(sma1))+1),
(Math.abs(getMax(sma2))-
sma2[i]+0.5)/(Math.abs(getMin(sma2))+Math.abs(getMax(sma2))+1),
"" + i);
                        };
                };

                if ((getMin(sma1)<0 & getMax(sma1)<0 ||
getMin(sma1)>0 & getMax(sma1)>0) & (getMin(sma2)<0 &
getMax(sma2)<0 || getMin(sma2)>0 & getMax(sma2)>0)){
                        double cost1;
                        for (int i = 0; i<sma1.length; i++){
                                cost1 = 0;
                                for (int j = 0; j<i; j++){
                                        if (sma1[i] == sma1[j] &
sma2[i] == sma2[j]){
                                                cost1 += 0.07;
                                        };
                                };

    plot.addLabel((Math.abs(getMin(sma1))+sma1[i]+0.5+cost1)/(-
Math.abs(getMin(sma1))+Math.abs(getMax(sma1))+1),
(Math.abs(getMax(sma2))-sma2[i]+0.5)/(-
Math.abs(getMin(sma2))+Math.abs(getMax(sma2))+1), "" + i);
                        };
                };

                plot.show();
```

```
    /**************************************************
*********/


                    double [] lma1 = new double [lmat[0].length];
                    double [] lma2 = new double [lmat[0].length];

                    for (int i = 0; i<lmat[0].length; i++){
                            lma1[i] = lmat[0][i];
                            lma2[i] = lmat[1][i];
                    }   //to plot


    /**************************************************
*********/

                    Plot plot1 = new Plot ("Variable loadings on
eigenvectors 1-2"+ totcum[1] +"% of total variance", "lma1", "lma2",
new double[]{0.0}, new double[]{0.0});
                    plot1.setLimits(getMin(lma1)-0.5,
getMax(lma1)+0.5, getMin(lma2)-0.5, getMax(lma2)+0.5);
                    plot1.setColor(Color.blue);
                    plot1.addPoints(lma1, lma2, Plot.CIRCLE);

    //plot1.addLabel((Math.abs(getMin(lma1)))/(Math.abs(getMin(lm
a1))+Math.abs(getMax(lma1))),
(Math.abs(getMax(lma2)))/(Math.abs(getMin(lma2))+Math.abs(getM
ax(lma2))), "+");

                    if (getMin(lma1)-0.5<0 & getMax(lma1)+0.5>0 &
getMin(lma2)-0.5<0 & getMax(lma2)+0.5>0 ){
                            double cost1;
                            for (int i = 0; i<lma1.length; i++){
                                    cost1 = 0;
                                    for (int j = 0; j<i; j++){
                                            if (lma1[i] == lma1[j] &
lma2[i] == lma2[j]){
                                                    cost1 += 0.07;
                                            };
                                    };

    plot1.addLabel((Math.abs(getMin(lma1))+lma1[i]+0.5+cost1)/(M
ath.abs(getMin(lma1))+Math.abs(getMax(lma1))+1),
(Math.abs(getMax(lma2))-
lma2[i]+0.5)/(Math.abs(getMin(lma2))+Math.abs(getMax(lma2))+1),
"" + i);
                            };
                    };
```

```
                    if (getMin(lma1)-0.5<0 & getMax(lma1)+0.5>0 &
(getMin(lma2)-0.5<0 & getMax(lma2)+0.5<0 || getMin(lma2)-0.5>0
& getMax(lma2)+0.5>0 )){
                        double cost1;
                        for (int i = 0; i<lma1.length; i++){
                            cost1 = 0;
                            for (int j = 0; j<i; j++){
                                if (lma1[i] == lma1[j] &
lma2[i] == lma2[j]){
                                    cost1 += 0.07;
                                };
                            };

    plot1.addLabel((Math.abs(getMin(lma1))+lma1[i]+0.5+cost1)/(M
ath.abs(getMin(lma1))+Math.abs(getMax(lma1))+1),
(Math.abs(getMax(lma2))-lma2[i]+0.5)/(-
Math.abs(getMin(lma2))+Math.abs(getMax(lma2))+1), "" + i);
                        };
                };

                    if ((getMin(lma1)-0.5<0 & getMax(lma1)+0.5<0 ||
getMin(lma1)-0.5>0 & getMax(lma1)+0.5>0) & getMin(lma2)-0.5<0
& getMax(lma2)+0.5>0 ){
                        double cost1;
                        for (int i = 0; i<lma1.length; i++){
                            cost1 = 0;
                            for (int j = 0; j<i; j++){
                                if (lma1[i] == lma1[j] &
lma2[i] == lma2[j]){
                                    cost1 += 0.07;
                                };
                            };

    plot1.addLabel((Math.abs(getMin(lma1))+lma1[i]+0.5+cost1)/(-
Math.abs(getMin(lma1))+Math.abs(getMax(lma1))+1),
(Math.abs(getMax(lma2))-
lma2[i]+0.5)/(Math.abs(getMin(lma2))+Math.abs(getMax(lma2))+1),
"" + i);
                        };
                };

                    if ((getMin(lma1)-0.5<0 & getMax(lma1)+0.5<0 ||
getMin(lma1)-0.5>0 & getMax(lma1)+0.5>0) & (getMin(lma2)-0.5<0
& getMax(lma2)+0.5<0 || getMin(lma2)-0.5>0 &
getMax(lma2)+0.5>0)){
                        double cost1;
                        for (int i = 0; i<lma1.length; i++){
                            cost1 = 0;
                            for (int j = 0; j<i; j++){
                                if (lma1[i] == lma1[j] &
lma2[i] == lma2[j]){
```

```
                                        cost1 += 0.07;
                                    };
                                };

    plot1.addLabel((Math.abs(getMin(lma1))+lma1[i]+0.5+cost1)/(-
Math.abs(getMin(lma1))+Math.abs(getMax(lma1))+1),
(Math.abs(getMax(lma2))-lma2[i]+0.5)/(-
Math.abs(getMin(lma2))+Math.abs(getMax(lma2))+1), "" + i);
                            };
                    };

                    plot1.show();
            };

            return lmat;
    };

    public static Zmat sqrtMatrix(double[][] matrics) {

            //double[][] root = new
double[matrics.length][matrics[0].length];

            int n = matrics.length; // matrics deve essere quadrata
            int n1 = matrics[0].length;

            if (n != n1) {
                    System.out.println("Matrix must be square!");
            }

            Zmat sqrtmatrics = new Zmat(matrics);

            try {
                    double[][] T = new Zmat(matrics).getRe();
                    double[][] U = new Zmat(n, n).getRe();
                    intW result = new intW(0);
                    int workLen = 8 * n;
                    org.netlib.lapack.DGEES.DGEES("V", "N", new
Object(), n, T,
                                    new intW(0), new double[n], new
double[n], U,
                                    new double[workLen], workLen,
new boolean[n], result);

                    // UTU* = matrics

                    Zmat scT = new Zmat(T);
                    Zmat scU = new Zmat(U);

                    Zmat R = new Zmat(scT.nr, scT.nc);
                    for (int i = 0; i < R.nr; i++)
```

```
                        R.put(i, i, scT.get(i, i)); // R è la diag(T) di
Matlab

                    if (R.equals(scT)) // scT è la T di matlab
                    {
                        for (int i = 0; i < R.nr; i++)
                            R.put(i, i, R.get(i, i).Sqrt());
                        // use diag
                    }
                    else {
                        R = new Zmat(R.nr, R.nc);
                        for (int j = 0; j < R.nc; j++)
                        {
                            R.put(j, j, scT.get(j, j).Sqrt());
                            for (int i = j - 1; i >= 0; i--) {
                                Zmat s = Times.o(R.get(i, i, i
+ 1, j - 1), R.get(i + 1, j - 1, j, j));
                                assert s.nc == 1 && s.nr ==
1;
                                R.put(i, j, scT.get(i,
j).Minus(s.get(0, 0)).Div(R.get(i, i).Plus(R.get(j, j))));
                            }
                        }
                    }

                    sqrtmatrics = Times.o(Times.o(scU, R),
Transpose.o(scU));
                    double[][] img = sqrtmatrics.getIm();
                    double[][] re = sqrtmatrics.getRe();

                    //root = re;

                } catch (JampackException e) {
                    e.printStackTrace();
                }

                return sqrtmatrics;
            }
            /* */


            /* Plots */
//          public void plotPCA(double [][] W, int cappa) { //param
necessari??? Triangolo
//
//              double xscale = 2/(Math.sqrt(3));
//
//              double [] x = new double [4];
//              x[0] = 0;
//              x[1] = xscale/2;
//              x[2] = xscale;
```

```
//              x[3] = 0;
//
//              double [] y = new double [4];
//              y[0] = -1;
//              y[1] = 0;
//              y[2] = -1;
//              y[3] = -1;
//
//              for (int i = 1; i<9; i++){
//                      double x1 = (i*xscale)/20;
//                      double x2 = ((20-i)*xscale)/20;
//                      double x3 = (i*xscale)/10;
//                      double y1 = -1+i/10;
//                      double y2 = -i/10;
//
//                      double [] xx = new double [2];
//                      xx[0] = x1;
//                      xx[1] = x2;
//
//                      double [] yy = new double [2];
//                      Arrays.fill(yy, y1);
//
//                      double [] xx2 = new double [2];
//                      xx2[0] = x1;
//                      xx2[1] = x3;
//
//                      double [] yy2 = new double [2];
//                      yy2[0] = y1;
//                      yy2[1] = -1;
//
//                      double [] xx3 = new double [2];
//                      xx3[0] = x3;
//                      xx3[1] = 0.5*xscale+x1;
//
//                      double [] yy3 = new double [2];
//                      yy3[0] = -1;
//                      yy3[1] = y2;
//
////                    Plot plotPCA = new Plot ("", "", "", x, y,
Plot.CIRCLE);
////                    plotPCA.setLineWidth(0);
////                    plotPCA.show();
//
//                      //CONTROLLA SE FUNZIONA
//                      Plot plotPCA = new Plot ("", "", "", new
double[]{0.0}, new double[]{0.0});
//                      plotPCA.setLimits(getMin(x), getMax(x),
getMin(y), getMax(y));
//                      plotPCA.addPoints(x, y, Plot.CIRCLE);
//                      plotPCA.show();
//              };
```

```
//    };
//
      public static void plotTwoComponents (double [][] L, int kappa,
String text){ //Scatter plot

            double [] x1 = new double [L.length];
            for (int i = 0; i<L.length; i++){
                  x1[i] = L[i][0];
            }

            double [] x2 = new double [L.length];
            for (int i = 0; i<x2.length; i++){
                  x2[i] = L[i][1];
            }

//          double x1max = x1[0];
//          for (int i = 0; i<x1.length-1; i++){
//                if (x1[i]<x1[i+1]){
//                      x1max = x1[i+1];
//                }
//          }
//
//          double x2max = x2[0];
//          for (int i = 0; i<x2.length-1; i++){
//                if (x2[i]<x2[i+1]){
//                      x2max = x2[i+1];
//                }
//          }
//
//          double x1min = x1[0];
//          for (int i = 0; i<x1.length-1; i++){
//                if (x1[i]>x1[i+1]){
//                      x1min = x1[i+1];
//                }
//          }
//
//          double x2min = x2[0];
//          for (int i = 0; i<x2.length-1; i++){
//                if (x2[i]>x2[i+1]){
//                      x2min = x2[i+1];
//                }
//          }

            Plot plot3 = new Plot(text, "Axis1", "Axis2", new
double[]{0.0}, new double[]{0.0});
            //plot3.setLimits(x1min-(0.05*(x1max-x1min)),
x1max+(0.05*(x1max-x1min)), x2min-(0.05*(x2max-x2min)),
x2max+(0.05*(x2max-x2min)));
            plot3.setLimits(getMin(x1)-0.5, getMax(x1)+0.5,
getMin(x2)-0.5, getMax(x2)+0.5);
            plot3.addPoints(x1, x2, Plot.CIRCLE);
```

```
		if (kappa == 1){
			plot3.setColor(Color.red);
		}

		if (kappa == 2){
			plot3.setColor(Color.green);
		}

		if (kappa == 3){
			plot3.setColor(Color.blue);
		}
		if (kappa != 1 & kappa != 2 & kappa != 3){
			plot3.setColor(Color.black);
		}

//		switch(kappa) {
//			case 1: plot3.setColor(Color.red); break;
//			case 2: plot3.setColor(Color.green); break;

//			case 3: plot3.setColor(Color.blue); break;
//			default: plot3.setColor(Color.black); break;
//		}

		if (getMin(x1)-0.5<0 & getMax(x1)+0.5>0 & getMin(x2)-
0.5<0 & getMax(x2)+0.5>0 ){
			double cost1;
			for (int i = 0; i<x1.length; i++){
				cost1 = 0;
				for (int j = 0; j<i; j++){
					if (x1[i] == x1[j] & x2[i] == x2[j]){
						cost1 += 0.07;
					};
				};

	plot3.addLabel((Math.abs(getMin(x1))+x1[i]+0.5+cost1)/(Math.a
bs(getMin(x1))+Math.abs(getMax(x1))+1), (Math.abs(getMax(x2))-
x2[i]+0.5)/(Math.abs(getMin(x2))+Math.abs(getMax(x2))+1), "" + i);
			};
		};

		if (getMin(x1)-0.5<0 & getMax(x1)+0.5>0 &
(getMin(x2)-0.5<0 & getMax(x2)+0.5<0 || getMin(x2)-0.5>0 &
getMax(x2)+0.5>0 )){
			double cost1;
			for (int i = 0; i<x1.length; i++){
				cost1 = 0;
				for (int j = 0; j<i; j++){
					if (x1[i] == x1[j] & x2[i] == x2[j]){
						cost1 += 0.07;
					};
```

```
                        };

    plot3.addLabel((Math.abs(getMin(x1))+x1[i]+0.5+cost1)/(Math.a
bs(getMin(x1))+Math.abs(getMax(x1))+1), (Math.abs(getMax(x2))-
x2[i]+0.5)/(-Math.abs(getMin(x2))+Math.abs(getMax(x2))+1), "" + i);
                    };
            };

            if ((getMin(x1)-0.5<0 & getMax(x1)+0.5<0 || getMin(x1)-
0.5>0 & getMax(x1)+0.5>0) & getMin(x2)-0.5<0 &
getMax(x2)+0.5>0 ){
                    double cost1;
                    for (int i = 0; i<x1.length; i++){
                            cost1 = 0;
                            for (int j = 0; j<i; j++){
                                    if (x1[i] == x1[j] & x2[i] == x2[j]){
                                            cost1 += 0.07;
                                    };
                            };

    plot3.addLabel((Math.abs(getMin(x1))+x1[i]+0.5+cost1)/(-
Math.abs(getMin(x1))+Math.abs(getMax(x1))+1),
(Math.abs(getMax(x2))-
x2[i]+0.5)/(Math.abs(getMin(x2))+Math.abs(getMax(x2))+1), "" + i);
                    };
            };

            if ((getMin(x1)-0.5<0 & getMax(x1)+0.5<0 || getMin(x1)-
0.5>0 & getMax(x1)+0.5>0) & (getMin(x2)-0.5<0 &
getMax(x2)+0.5<0 || getMin(x2)-0.5>0 & getMax(x2)+0.5>0)){
                    double cost1;
                    for (int i = 0; i<x1.length; i++){
                            cost1 = 0;
                            for (int j = 0; j<i; j++){
                                    if (x1[i] == x1[j] & x2[i] == x2[j]){
                                            cost1 += 0.07;
                                    };
                            };

    plot3.addLabel((Math.abs(getMin(x1))+x1[i]+0.5+cost1)/(-
Math.abs(getMin(x1))+Math.abs(getMax(x1))+1),
(Math.abs(getMax(x2))-x2[i]+0.5)/(-
Math.abs(getMin(x2))+Math.abs(getMax(x2))+1), "" + i);
                    };
            };

            plot3.show();

    };
    /* */
}
```

Appendice B
Algoritmo, implementato in ambiente Matlab, per la registrazione delle immagini

```
%%algoritmo di merge per visualizzare il claustro
%%ipotizzo che tutte le immagini siano in formato bitmap
e che per
% REGISTERING AN IMAGE - help

%modus operandi
%primo step: acquisizione immagini e passaggio a
immagini a livelli di grigio;
%secondo step: bilanciamento livelli di grigio
%terzo step: registrazione immagini

image1 = imread('im1.bmp');
image1 = rgb2gray(image1);

image2 = imread('im2.bmp');
image2 = rgb2gray(image2);

%%%SE NECESSARIO, SI PROCEDE CON UNA CAMPITURA DEL
BACKGROUND
%background = imopen(image1, strel('disk', 150));
%image1 = imsubtract(image1, background);e1
%image1 = imadjust(image1, stretchlim(imag1));

%background = imopen(image2, strel('disk', 150));
%image2 = imsubtract(image2, background);
%image2 = imadjust(image2, stretchlim(due1));

cpselect(image2, image1);

%export three points to a workspace
    %%input_points sono relativi a imag,e2 mentre
base_points sono relativi a
    %%image1

%%% PROCEDO CON IL MERGE
tform = cp2tform (input_points, base_points, 'affine');
[registered2 xdata ydata] = imtransform (image2, tform,
'FillValues', 255);
```

```
figure, imshow ( registered2, 'XData', xdata, 'YData',
ydata);
hold on
h = imshow(image1, gray(256));
ylim = get(gca, 'YLim');
set(gca, 'YLim', [0 ylim(2)]);

figure
parz = imshow(imread('parz.jpg'));
parz = imcrop;
parz = rgb2gray(parz);
imwrite (parz, 'im.bmp') %immagine salvata come 1e2, in
formato .bmp
delete parz.jpg
```

Bibliografia e Sitografia

1. J. R. Glaser, E. M. Glaser “Neuron imaging with Neurolucida – a pc-based system for image combining microscopy” Computerized Medical Imaging and Graphics, Vol.14, Issue 5, Settembre 1990, pagg. 307-317;
2. K. M. Brown, D. E. Donohue, G. D’Alessandro, G, A, Ascoli “A Cross-Platform Freeware tool for Digital Reconstruction of Neuronal Arborizations From Image Stacks”, Neuroinformatics, Vol.3, No. 4, 2005, pagg. 343-359;
3. www.mbfbioscience.com/neurolucida;
4. S. K. Schmitz, J.J.J. Hjorth, R. M. S. Joemai, R. Wijntjes, S. Eijgeraam, P. de Bruijn, C. Georgiou, A. P. H. de Jong, A. van Ooyen, M. Verhage, L. N. Cornelisse, R. F. Toonen, W. Veldkamp “Automated analysis of neuronal morphology, synapse number and synaptic recruitment”, Journal of Neuroscience, Novembre 2011, pagg. 185-193;
5. http://imagej.nih.gov/ij/docs/guide/userguide-1.html#toc-Section-1;
6. www.imagescience.org/meijering/software/neuronj
7. E. Meijering, M. Jacob, J. C. F. Sarria, P. Steiner, H. Hirling, M. Unser “Design and validation of a tool for Neurite Tracing and Analysis in Fluorescence Microscopy Images”, Cytometry Part A, Vol. 58, No. 2, April 2004, pagg. 167-176;
8. http://fournierlab.mcgill.ca/neuritetracer.html
9. M. Pool, J. Thiemann, A. Bar-Or, A.E Fournier “Neurite Tracer: A novel ImageJ plugin for automated quantification of neurite outgrowth”, Journal of Neuroscience Methods, Vol. 168, Issue 1, 2007, pagg. 134-139;
10. S. Y. Ho, C.Y. Chao, H.L. Luang, T.W. Chiu, P. Charoenkwan, E. Hwang “NeurphologyJ: An automatical neuronal morphology

quantification method and its application in pharmacological discovery” , Bioinformatics, Giugno 2011, pagg. 2-18;

11. M. L. Narro, F. Yang, R. Kraft, C. Wenk, A. Efrat, L. L. Restifo “NeuronMetrics: Software for semi.automated processing of cultured-neuron Images”, Brain Res., Marzo 2008, pagg. 57-75;
12. http://imagejdocu.tudor.lu/doku.php?id=plugin:analysis:asa:start
13. http://rsbweb.nih.gov/ij/plugins/fraclac/FLHelp/Introduction.htm
14. http://en.wikipedia.org/wiki/Neuronstudio
15. S.L.Wearne, A. Rodriguez, D.B. Ehlenberger, A.B. Rocher, S.C. Henderson, and P.R. Hof., “New techniques for imaging, digitization and analysis of three-dimensional neuronal morphology on multiple scales”, *Neuroscience*, 2005, pagg. 661-680;
16. A. Rodriguez, D. B. Ehlenberg, D. L. Dickstein, P. R. Hof, S. L. Wearne “Automated Three-Dimensional Detection and Shape Classification of Dendritic Spines from Fluorescence Microscopy Images”, PloS ONE, Vol.3, Issue 4, Aprile 2008, pagg. 1-12;
17. J. S. Gensel, D. L. Schonberg, J. K. Alexander, D. M. Mc Tique, PG Popovich “Semiautomated sholl analysis for quantifying changes in growth and differentiation of neurons and glia” , Journal of Neuroscience Methods, Giugno 2010, pagg. 71-79;
18. http://www.moleculardevices.com/Products/Software/Meta-Imaging-Series/MetaMorph.htm
19. H. Cuntz, F. Forstner, A. Borst, M. Hausser “One rule to grow them all: a General Theory of Neuronal Branching and its Pratical Application”, Computational Biology, Vol. 6, Issue 8, Agosto 2010, pagg.1-14;
20. http://www.treestoolbox.org/manual.html;
21. H. Cuntz, F. Forstner, A. Borst, M. Haussner “The TREE- toolbox: probing the basis of axonal and dendritic branching”, Neuroinform, January 2011, pagg. 91-96;
22. T. L. Kemper, M. L. Baumann, “Neuropathology of infantile autism”, Journal of Neuropathology and Experimental Neurology, Vol 57, No. 7, pagg. 645-656;

23. M. L. Baumann, T. L. Kemper, “Neuroanatomics observations of the brain in autism: a review and future directions”, International Journal of Developmental Neuroscience, Settembre 2004, pagg. 183-187;
24. E. Di Cicco-Bloom, C. Lord, L. Zwaigenbaum, E. Courchesne, S. R. Dager, C. Schmiltz, R. T. Schultz, J. Crawley, L. J. Young “The developmental Neurobiology of Autism Spectrum disorder”, The Journal of Neuroscience, Giugno 2006, pagg. 6897-6906;
25. M. L. Baumann, “Microscopic Neuroanatomic Abonormalities in autism”, Pediatrics, Vol. 87, No. 5., Maggio 1991, pagg. 791-796;
26. D. H. Geschwind, P. Levitt, “Autism spectrum disorder: developmental disconnection syndromes”, Nuerobiology, Vol. 17, 2007, pagg. 103-111;
27. E. Courchesne, C. : Karns, H. R. Davis, R. Ziccardi, IR. A. Carper, Z. D. Tigue, H. J. Chisum, P. Moses, K. Pierce, C. Lord, A. J. Lincoln, S. Pizzo, L. Schreibman, R. H. Haas, N. A. Akshoomoff, R. Y. Courchesne, “Unusual brain growth patterns in early life in patients with autistic disorder”, Neurology, Vol. 57, No. 2, Luglio 2001, pagg. 245-254;
28. E. Courchesne, “Brain development in autism: early overgrowth followed by premature arrest of growth”, Developmental Research Reviews, Vol. 10., Issue 2, Maggio 2004, Pag. 106-111;
29. M. F. Casanova, D. P. Buxhoeveden, A. E. Switala, E. Roy, “Neuronal density and architecture in the brains of autistic patients”, Neurology, Vol. 17, No. 7, Luglio 2002, pagg. 505-521;
30. ER Ritvo, “Lower Purkinje cell counts in the cerebella of four autistic subject: initial findings of the UCLA-N SAC autopsy research”, The American Journal of autism, Luglio 1986, pagg. 862-866;
31. E. R. Whitney, T. L. Kemper, M. L. Baumann, D. L. Rosme, G. J. Blatt, “Cerebellar Purkinje cells are reduced in a subpopulation of autistic brain: a stereological experiment using Calbidin-d28k”, Cerebellum, Vol.7, No. 3, 2008, pagg. 406-416;
32. A. Bailey, P. Luthert, A. Dean, B. Harding, I. Janota, M. Montgomery, M. Rutter, “A clinicopathological study of autism”, Brain, 1998, pagg. 889-905;

33. S. H. Fatemi, A. R. Halt, G. Realmut, J. Earle, D. A. Kist, P. Thuras, A. Merz, “Purkinje cell size is reduced in cerebellum of patients with autism”, Cellular and Molecular Neurobiology, Vol. 22, No.2, Aprile 2002, pagg. 171-175;
34. J. Wegiel, I. Kuchna, K. Nowiki, H. Imaki, J. Wegiel, E. Marchi, S. Y. Ma, A. Chauhan, V, Chauhan, T. Wierzba Bobrowics, M. de Leon, L. A. Saint Louis, I. L. Cohen, E. London, W. T. Brown, T. Wisniewski “The neuropathology of autism: defects of neurogenesis and neuronal migration, and dysplastic changes”, Acta Neuropathologica, Febbraio 2010, pagg. 755-770;
35. it.wikipedia.org/wiki/claustro;
36. W. B. Davis, “The claustrum in autism and typically developing. Male children: a quantitative MRI study”, Tesi di laurea, Dicembre 2008;
37. R. Brookmeyer, S. Gray, c. Kawas “Projections of Alzheimer’s disease in the United States and the public health impact od delaying disease onset”, American Journal of public health, Settembre 1998, Vol. 88, No. 9, pagg. 1337-1342;
38. R. Brookmeyer, E. Johnson, K. Ziegler-Graham, H.M. Arrighi “Forecasting the global burden of Alzheimer’s disease”, Alzheimer & Dementia: the journal of Alzheimer’s association, Vol. 3, Issue 3, Luglio 2007, pagg. 186-191;
39. http://it.wikipedia.org/wiki/Morbo_di_Alzheimer
40. G. McKhann, D. Drachman, M. Folstein, R. Katzman, D. Price, E. M. Stadlan “Clinical diagnosis of Alzheimer’s disease: report of the NINCOS-ADRDA work group under the auspices of Department of Health and Human Service Taskforce on Alzheimer’s disease”, Neurology, Luglio 1984, Vol. 34, pagg. 939-945;
41. G. Ricci, L. Volpi, L. Pasquali, L. Petrozzi, G. Siciliano, “Astrocytes-Neuron interactions in neurological disorders”, Journal of Biological Physics, Vol. 35, Aprile 2009, pagg. 317-336;
42. P. J. Whitehouse, D. L. Price, R. G. Struble, A. W. Clark, J. T. Coyle, M. R. Delong, “Alzheimer’s disease and senile dementia: loss of neurons in the basal forebrain”, Science, Marzo 1982, Vol. 215, No. 4357, pagg. 1237-1239;

43. P. J. Whitehouse, D. L. Price, A. W. Clark, J. T. Coyle, M. R. Delong, “Alzheimer’s disease: evidence for selective loss of cholinergic neurons in the nucleus basalis”, Annals of Neurology, Vol. 10, Issue 2, Ottobre 2004, pagg. 122-126;
44. C. B. Saber, D. C. German, C. L. White, “Neuronal pathology in the nucleus basalis and associated cell groups in senile dementia of tha Alzheimer’s type: possible role in cell loss”, Neurology, Agosto 1985, pagg. 1089-1095;
45. T. Arendt, V. Bigl, A. Arendt, A. Tennstedt, “Loss of neurons in the nucleus basalis of Meynert in Alzheimer’s disease, paralysis and Korsakoff’s disease”, Acta Neuropathologica, 1983, Vol. 61, No. 2, pagg. 101-108;
46. T. Arendt, V. Bigl, A. Tennstedt, A. Arendt, “Neuronal loss in different parts of the nucleus basalis is related to neuritic plaque formation in cortical target areas in Alzheimer’s disease”, Neuroscience, Gennaio 1985, pagg. 1- 14;
47. K. M. Cullen, G. M. Halliday, K. L. Double, W. S. Brooks, H. Creasey, G. A. Broe, “Cell loss in the nucleus basalis is related to regional cortical atrophy in Alzheimer’s disease”, Neuroscience, Giugno 1997, Vol. 78, pagg. 641-652;
48. S. J. Allen, D. Dawborn, G. K. Wilcock, “Morphometric immunochemical analysis of neurons in the nucleus basalis of Meynert in Alzheimer’s disease”, Vol. 254, Giugno 1988, pagg. 275-281;
49. P. D. Coleman, D. G. Flood, “Neuron numbers and dendritic extent in normal aging Alzheimer’s disease”, Neurobiology oh aging, Vol. 8, Issue 6, Dicembre 1987, pagg. 521-545;
50. V. C. Palay, E. Asan, “Alterations in catecholamine neurons of the locus coeruleus in senile dementia of the Alzheimer type and in Parkinson’s disease with or without dementia and depression”, Journal of Comparative Neurology, Vol. 287, Issue 3, Settembre 1989, pagg. 373-392;
51. S. Knafo, C. Venero, P. Merino-Serrais, I. Fernand-Espinoza, J. Gonzalez-Soriano, I. Ferrer. G. Santpen, J. Defelipe, “Morphological

alterations of neurons of the amygdala and impaired fear conditioning in a transgenic mouse model of Alzheimer's disease", The Journal of Neuroscience, Settembre 2009, pagg. 41-51; ù

52. J. Hu, K. T. Akama, G. A. Krafft, B. A. Chromy, L. J. Van Eldik, "Amyloid-β peptide activates cultured astrocytes: morphological alterations cytokine induction and nitric oxide release", Brain Research, Vol. 785, Marzo 1998, pagg. 195-206;
53. http://it.wikipedia.org/wiki/Malattia_idiopatica_di_Parkinson;
54. H. Braak, K. Del Tredici, U. Rub, R. A, I. de Vos, E. N. H. Jansen Steur, E. Braak, "Staging of brain pathology related to sporadic Parkinson's disease", Neurobiology of aging, Vol. 24, Issue 2, Marzo 2003, pagg. 197-211;
55. K. Jellinger, "Rapporti neuropatologici tra Parkinson e demenze";
56. J. M. Candy, R. H. Perry, E. K. Perry, D. Irving, G. Blened, A. F. Fairbain, B. E. Tomlinson, "Pathological changes in the nucleus of Meynert in Alzheimer's and Parkinson's disease", Journal of Neurology, Maggio 1983, Vol. 59, pagg. 277-289;
57. I. Nakamo, A. Hirano, "Parkinson's disease: neuron loss in the nucleus basalis without concomitant Alzheimer's disease", Annals of Neurology, Vol. 15, Issue 5, Maggio 1984, pagg. 415-418;
58. P.Damier, E. C. Hirsch, Y. Agid, A. M. Graybell, "The substantia nigra of the human brain. Pattern of loss of domanine-containing neurons in Parkinson's disease", Journal of Neurology, 1992, Vol. 122, pagg. 1437-1448;
59. J. O. Rinne, J. R. Mlic, L. Paljärvi, U. K. Rinne, "Dementia in Parkinson's disease is related to neuronale loss in the medial substantia nigra", Annals of Neurology, Vol. 26, Issue 1, Giugno 1989, pagg. 47-50;
60. G. M. Halliday, P. C. Blumbergs, R. G. H. Cotton, W. W. Blessing, L. B. Geffen, "Loss of brainsteim serotonin- and substance P-containing neurons in Parkinson's disease", Brain Research, Vol. 510, Issue 1; Febbraio 1990, Pagg. 104-107;
61. O. Solis, D. I. Limon, J.Flores-Hernandez, G. Flores "Alterations in dendritic morphology of the prefrontal cortical and striatum neurons

in the unilateral 6-OHDA rat model of Parkinson's disease", Synapse, Vol. 61, Issue 6, Giugno 2007, pagg. 450-458;

62. http://it.wikipedia.org/wiki/Malattia_di_Huntington;
63. http://www.retemalattierare.it/modules.php?name=News&file=print&sid=2110;
64. X. Gu, E. R. Greiner, R. Mishra, R. Kodali, A. Osmand, S. Finkbeiner, J. S. Steffan, L. M. Thompson, R. Wetzel, X. W. Yang, "Serine 13 and 16 are critical determinants of full-length human mutant Huntington induced disease pathogenesis in HD mice", Neuron, Vol. 64, Issue 6, Dicembre 2009, pagg. 828-840;
65. J. C. Hedreen, C. L Peyser, S. E. Folstein, C. A. Ross, "Neuronal loss in V and VI of cerebral cortex in Huntington's disease", Neuroscience letters, Vol. 133, Issue 2, Dicembre 1991, pagg. 257-261;
66. G. Rajkowska, L. D. Selemon, P. S. Goldman-Rakic, "Neuronal and glial somal size in the prefrontal cortex: a postmortem morphometric study of schizophrenia and Huntington's disease", Archives of general psychiatry, Vol. 55, 1998, pagg.215-224;
67. E. K. Richfield, K. A. Maguire-Zeiss, H. E. Vonkeman, P. Voorn, "Preferenzial loss of prepoenkephalin versus preprotachykinin neurons from the striatum of Huntington's disease patients", Annals of Neurology, Vol. 38, Issue 6, Dicembre1995, pagg. 852-861;
68. E. J. Slow, J. van Raamsdonk, D. Rogers, S. H. Coleman, R. K. Graham, Y. Deng, R. Oh, N. Bissada, S. M. Hossain, Y. Yang, X. Li, E. M. Simpson, C. Gatekunst, B. R. Leavitt, M. R Hayden, "Selective striatal neuronal loss in a YAC128 mouse model of Huntington's disease", Human Molecular Genetics, Vol. 12, 2003, pagg. 1557-1567;
69. G. L Klapstein, R. S. Fisher, H. Zanjam, C. Cepeda, E. S. Jokel, M. F. Chesselet, M. S. Levine, "Electrophysiological and morphological changes in striatal spiny neyrons in R6/2 Huntington's disease transgenic mice", Journal of Neurophisiology, Maggio 2001, Vol. 86, No.6, pagg. 2667-2674;
70. Z. C. Baquet, J. A. Gorski, K. R. Jones., "Early striatal dendritic deficits followed by neuron loss in advanced age in the absence of

anterograde cortical brain-derived neurotrophic factor", The Journal of Neuroscience, Aprile 2004, pagg. 4250-4258;

71. R. P. Lerner, L. D. Trejo-Martinez, C. Zhu, M. F. Chesselet, M. A. Hickey, "Striatal atrophy and dendritic alterations in a knock-in mouse model of Huntington disease", Brain Research Bullettin, Febbraio 2012 (in press);
72. T. L. Spires, H. E. Grate, S. Garry, P. M. Cordery, A. van Dellen, C. Blakemore, A. J. Hannan, "Dendritic spine pathology and deficits in experience-dependent dendritic plasticity in R6/1 Huntington's disease transgenic mice", European Journal of Neuroscience, Vol. 19, Maggio 2009, pagg. 2799-2807;
73. R. Martinez-Tellez, M. Gomez-Villalobos, G. Flores, "Alteration in dendritic morphology of cortical neurons in rats with diabetes mellitus induced by streptozotocin", Brain Research, Vol. 1048, Issue 2, Giugno 2005, pagg. 108-115;
74. J. N. Riley, D. W. Walker, "Morphological alterations in hippocampus after-long term alcohol consumption in mice", Science, Vol. 201, No. 2043, pagg. 646-648;
75. A. Ruiz-Marcus, F. Sanchez-Toscano, F. Escobar, D. Rey, G. Morreale de Escobar, "Reversible morphological alterations of cortical neurons in juvenile and adult hypothyroidism in the rat", Brain Research, Marzo 1980, Vol. 185, pagg. 91-102;
76. S. U. Walkley, M. E. Haskins, R. M. Shull, "Alterations in neuron morphology in mucopolysaccharidosis type 1", Acta Neuropathologica, 1988, Vol. 75, No.6, pagg. 611-620;
77. T. E. Robinson, B. Kolb, "Alteration in the morphology of the dendrites and dendritic spines in the nuncleus accumbens and prefrontal cortex following repeated treatment with amphetamine or cocaine", European Journal of Neuroscience, Maggio 1999, Vol. 11, Issue 5, pagg. 1598-1604;
78. C.Magliaro, "Ne.Mo.: sviluppo di un'interfaccia grafica per l'elaborazione e l'analisi automatizzata di neuroni", tesi di laurea triennale in Ingegneria Biomedica, Pisa 2009;

79. L. Billeci, “Analisi della connettività e della dinamica di crescita di cellule di Purkinje tramite imaging in vitro ed algoritmi computazionali”, tesi di laurea specialistica in Ingegneria Biomedica, Pisa 2007;
80. M. Brancadoro “Confronto tra cellule di Purkinje estratte da topi wild-type e da modelli animali di autismo tramite un tool di analisi morfologica”, tesi di Laurea Triennale in Ingegneria Biomedica, Maggio 2011;
81. H. A. L. Kiers, I. Van Groningen, “Three-way component analysis: principles and illustrative applications”, Psychological Methods, 2001;
82. *Titolo*: NEMO: Neuron Morphological Analysis Tool 1.0
Autori: Billeci Lucia, Magliaro Chiara, Giovanni Pioggia, Arti Ahluwalia
Data registrazione: 13/12/2011;
Numero progressivo: 008241;
Ordinativo: D007440
83. M. Sciabarrà, “Manuale Java”;
84. M. Tarquini, A. Ligi, “Java mattone dopo mattone”, pagg. 33-90;
85. G. Grosso, “Istallazione di Java-Uso delle librerie”;
86. http://www.dis.uniroma1.it/~monscan/Didattica/MaterialeDidattico/ProgSw0203/SecondaLezione/uni7.html;
87. E. Collnot, B. Gaiser, N. Ucciferri, A. Ahluwalia, “Protocols to determine cytotoxicity oxidative stress and infiammatory response to NPs”, 10 Ottobre 2011.

Ringraziamenti

Un doveroso ringraziamento a quanti mi hanno sostenuto.
Ringrazio la prof.ssa Arti Ahluwalia, che ha creduto nelle mie potenzialità. Mi ha permesso di imparare un nuovo linguaggio di programmazione, e di appassionarmi a un nuovo campo di ricerca, che per paura ho sempre considerato lontano dalle mie capacità.
Un doveroso ringraziamento al prof. Riccardo Leardi, non solo per la possibilità di tradurre i suoi algoritmi, ma anche per il materiale fornitomi e per la cordialità di mostratami fin dalle prime mail.
Un grazie di cuore anche a Nicole Lazzeri, che con somma pazienza mi ha seguito nei primi passi di implementazione del plug-in, nella costruzione del .jar, e ha sopportato le mie innumerevoli mail a qualsiasi ora del giorno.
Ringrazio inoltre Nadia Ucciferri, che ha messo a disposizione parte dei suoi dati della tesi di dottorato, e Rita Antonini, che ha preparato i vetrini di claustro che ho fotografato durante il mio lavoro di tesi.
Ultima, ma non per importanza, la persona che mi è stata vicina in questi mesi, Lucia Billeci: ha saputo sempre come spronarmi, mi ha dato la possibilità di brevettare con lei Ne.Mo., diventando oltre che la mia tutor, la mia confidente accademica.