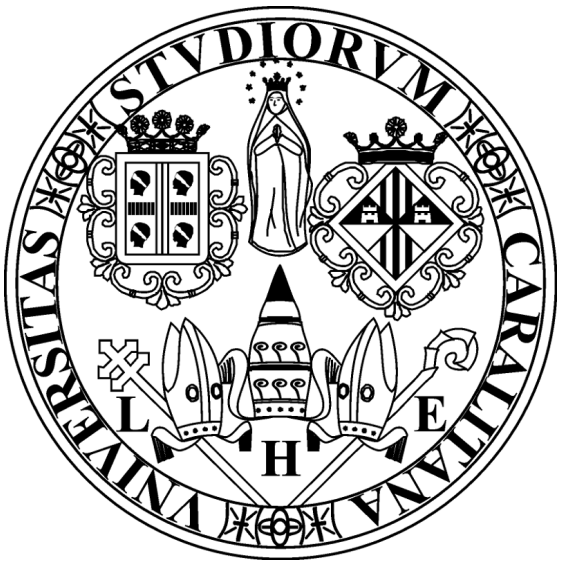



# Università degli studi di Cagliari

## Facoltà di Ingegneria

*Corso di laurea triennale in Ingegneria Elettronica*

## Algoritmi Ottimizzati per la Photometric Stereo Applicati all'Archeologia

**Relatore:**

*Prof. Giuseppe Rodriguez*

**Controrelatore:**

*Prof. Massimo Vanzi*

**Candidato:**

*Riccardo Dessì*

**SESSIONE UNICA A.A. 2012/2013**

*Dedico questo testo alla mia famiglia e a tutti coloro che in questi 3 anni hanno avuto la pazienza di sopportarmi e volermi bene (il che non è una cosa semplice).*

# Indice

## 4 Risultati numerici

# Prefazione

L'obbiettivo di questo lavoro è la: *Photometric Stereo.* Ovvero è una tecnica che consente, in base modelli matematici dedicati, la ricostruire delle superfici 3-D, a partire semplicemente da delle fotografie acquisite secondo condizioni di illuminazione differenti. Rispetto ad altre metodologie di ricostruzione (ad esempio scanner), la Photometric Stereo risulta molto economica e facile da implementare. Per questo motivo la sua applicazione trova ampio riscontro in ambito archeologico. Infatti una necessità culturale forte di questi ultimi tempi, è proprio quella di raccogliere e catalogare, all'interno di grosse banche dati digitali, reperti come: rilievi, incisioni rupestri ecc, che con l'andare del tempo ovviamente sono soggetti all'erosione degli agenti atmosferici. Non solo, ma Il fatto di avere a disposizione delle ricostruzioni fedeli 3-D di tali manufatti consente una migliore analisi a posteriori da parte degli archeologi che hanno la possibilità di visualizzare gli oggetti sotto punti di vista differenti. Ovviamente però tale processo deve fare fronte a delle esigenze di budget limitati e di praticità, nella misura in cui i reperti sono situati in aree che non si prestano all'utilizzo di attrezzatura ingombrante e costosa. Il nostro metodo a tal proposito, supera questi limiti consentendo di ottenere ottimi risultati in relazione alla semplicità e praticità di utilizzo. La Photometric Stereo nacque approssimativamente quindici anni fa, e si sviluppò particolarmente a seguito delle grandi innovazioni tecnologiche in ambito *image processing* e matematico. Uno dei problemi che però ha sempre posto delle criticità importanti era sicuramente l'aspetto dei tempi di calcolo dovuto al processo di elaborazione numerica. Pertanto questo lavoro si pone come obbiettivo cardine quello di studiare e analizzare tale fenomeno cercando di sintetizzare degli algoritmi ottimizzati in accordo con tutto quello che è già stato sviluppato presso il dipartimento di ingegneria elettronica degli studi di Cagliari, in particolare dall' ing. Pintus e dal Prof. Vanzi. Particolari ringraziamenti vanno al Prof. Giuseppe Rodriguez, che in questi mesi si è interessato particolarmente al problema ed è sempre stato in grado di indirizzarmi verso orizzonti che hanno consentito la riuscita di questo lavoro. Ulteriori sentiti ringraziamenti vanno al Prof. Massimo Vanzi e la Dott.ssa Carla Mannu, che si sono resi partecipi fornendomi il materiale necessario per lo sviluppo della parte sperimentale e supervisionando il lavoro fornendomi interessati spunti di riflessione che hanno arricchito il lavoro svolto.

# 1 Introduzione alla Photometric Stereo

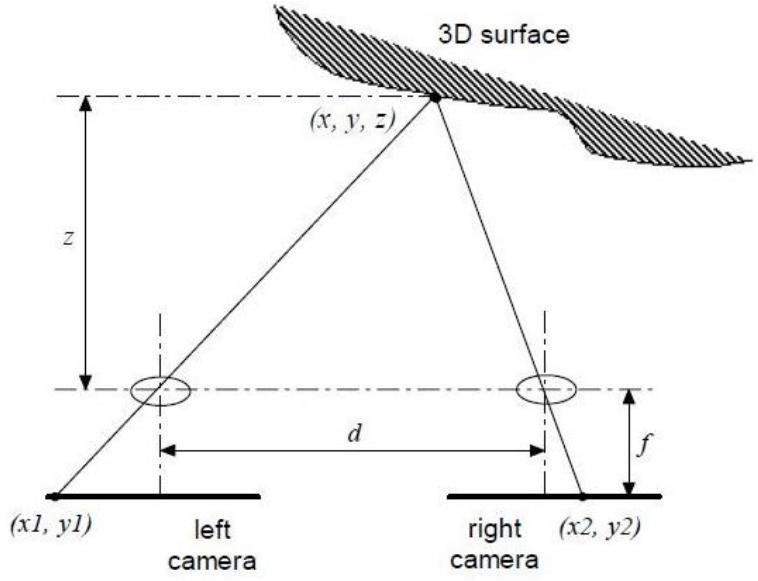
## 1.1 Cosa si intende per fotometria

La fotometria è una tecnica che consente di analizzare il flusso della radiazione elettromagnetica di un oggetto. Principalmente viene utilizzata in campo astronomico, ad esempio, per determinare l'età di una stella e quindi lo stato della suo ciclo vitale. In realtà quest'ultima trova sede in una miriade di problemi ingegneristici attuali. Motivo per il quale, In questo testo, ricorreremo alla fotometria per affrontare uno dei temi caldi dell'image processing moderno. Ovvero a partire da una serie di fotografie, il nostro obbiettivo sarà quello di ricostruire le superfici 3-D degli oggetti che compaiono nei vari frame. D'altronde una delle informazioni che si perde durante l'acquisizione di un'immagine è proprio la profondità che in realtà può essere ricostruita a partire dalla conoscenza delle caratteristiche fotometriche dell'oggetto, ad esempio lo spettro di radiazione. Tale problema può essere affrontato utilizzando due metodologie differenti meglio note come: *fotometria binoculare* o *Photometric Stereo.* Nel nostro caso faremo riferimento solamente alla seconda, nonostante ciò riportiamo di seguito una breve descrizione di entrambe le metodiche.

## 1.2Fotometria binoculare

La fotometria binoculare ha come finalità quella di ricavare la profondità di un oggetto a partire da una serie di fotografie scattate da diverse angolazioni. Questo si traduce in un problema di *corrispondenza* o *Stereo matching* , in altri termini, immaginando di avere solo due scatti s1 ed s2, ogni singolo punto di s1 deve trovare una corrispondenza biunivoca su s2 e questo altrettanto deve valere per tutti i punti di s2. Una volta ricreata una *mappa di corrispondenze* sarà possibile ricostruire la superficie 3-D dell'oggetto utilizzando la relazione che fa riferimento all'immagine sottostante:

$$z = \frac{df}{x1 - x2}$$



Come si evince dalla relazione, il numero minimo di frame per poter ricavare la profondità dell'oggetto deve essere almeno pari a due. In realtà se si vogliono ottenere buoni risultati il numero delle fotografie deve essere decisamente superiore e questo implica una serie di problemi. In primo luogo è necessario ricostruire per ogni frame una mappa di corrispondenza con le n immagini acquisite, maggiore sarà n  maggiore sarà il carico computazionale, ma ciò non è sufficiente a determinare il problema. Ciò dipende particolarmente anche dal modo con cui vengono acquisite le immagini, operazione non del tutto banale. Infatti questo presuppone la presenza di n camere per ogni punto di osservazione che deve essere scelto opportunamente, in quanto piccoli errori sulla posizione possono provocare problemi di indeterminazione durante la fase di corrispondenza.

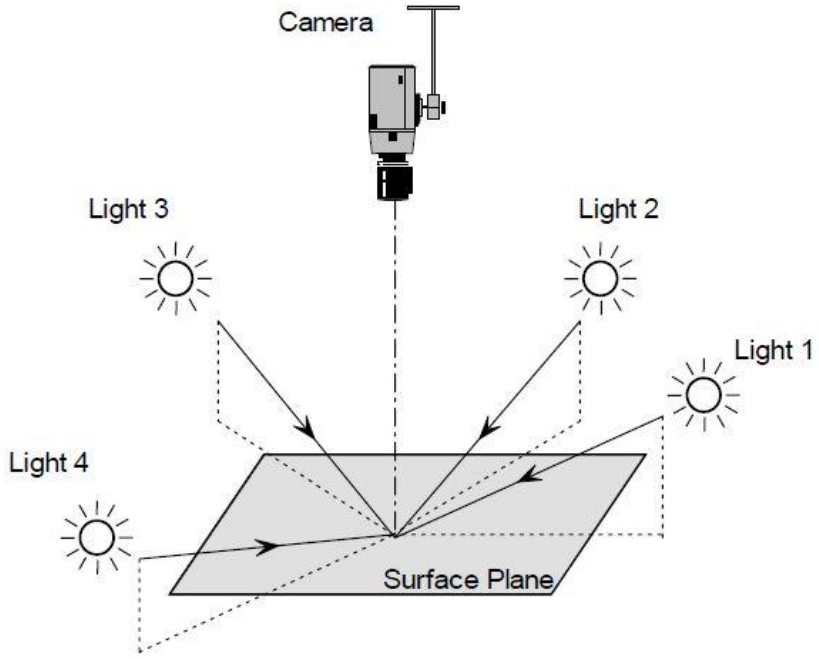

Nonostante tutta una serie di importanti precauzioni, sia pratiche che implementative,  (non di interesse dell'attuale trattazione), la fotometria binoculare ci consente di ricostruire le superfici 3-D degli oggetti in modo molto accurato e preciso. Motivo per il quale tale tecnica è utilizzata  in campo militare e specialmente in ambito topografico per la ricostruzione di complesse piante tridimensionali. Parte di questi risultati sono apprezzabili e già fruibili dagli utenti del web, come quanto già offerto da Google Maps, che recentemente ha realizzato un lavoro di questo tipo utilizzando immagini satellitari.

E' altresì vero che tecniche di questo genere risultano essere particolarmente costose motivo per il quale nel corso degli anni si sono sviluppate delle tecniche altrettanto efficaci e più riproducibili a livello industriale. Un esempio a tal proposito è proprio rappresentato dalla Photometric Stereo che ci accingiamo a descrivere.

## 1.3 Photometric Stereo

La Photometric Stereo, come detto precedentemente, nacque soprattutto come alternativa a metodi fotometrici molto costosi. In questo caso il punto di osservazione è sempre il medesimo e ciò presenta un grosso vantaggio sia in termini di costo, perché in questo caso è sufficiente una solo fotocamera, ma anche computazionali nella misura in cui non sussiste il problema della corrispondenza tipico della fotometria binoculare. Ogni singolo frame viene infatti acquisito illuminando gli oggetti, che si vogliono ricostruire, da diverse angolazioni. In altri termini in questa circostanza è la sorgente luminosa che viene spostata intorno all'oggetto. La finalità è dunque quella di studiare il comportamento fotometrico dell'oggetto, eccitato in condizioni differenti, ricostruendo una mappa dei gradienti e una mappa delle normali che ci consentono di stabilire

l'orientazione punto per punto dell'oggetto che può essere poi in un secondo momento utilizzata per ricostruire la superficie stessa. Di seguito riportiamo un illustrazione grafica del processo di acquisizione.




Questo tipo di metodica ha suscitato particolare interesse soprattutto in ambito archeologico. Si pensi infatti a un rilievo incastonato sul terreno, come fossili, o ancora dei simboli rupestri scolpiti nella roccia. In questo caso l'adozione di un processo di acquisizione che fa riferimento a punti di osservazione differenti risulta piuttosto complesso poiché spesso i siti archeologici si trovano in luoghi angusti e poco adatti a un sistema di acquisizione esteso. Per contro con la Photometric Stereo questo processo non solo risulta meno invasivo ma anche più rapido. Le motivazioni principali che dunque ci hanno suggerito di scegliere tale metodica, non sono solo puramente di tipo economico ma specialmente operativo. Infatti i problemi di Photometric Stereo possono essere affrontati in modo molto più semplice e sono molto più facili da implementare e assicurano tempi di calcolo del tutto ragionevoli, confrontati ad esempio a metodi binoculari.

## 1.4 Fattori di non idealità

In realtà esistono tutta una serie di limiti piuttosto importanti che a volte incidono pesantemente nella ricostruzione 3-D delle superfici. Prima fra tutte le proprietà fisiche degli oggetti, infatti per ottenere buoni risultati, quest'ultimi dovrebbero essere il più regolare possibili, quindi caratterizzati da geometrie non troppo complesse. A tal proposito si definiscono *Lambertiani* tutti quei corpi che rispettano la legge di *Lambert*, meglio nota come la *legge del coseno*. Quest'ultima afferma che l'illuminamento prodotto da una sorgente su una superficie è direttamente proporzionale all'intensità luminosa della sorgente e al coseno dell'angolo che la normale alla superficie forma con

la direzione dei raggi luminosi. In altri termini una superficie può essere definita *Lambertiana,* se indipendentemente dal punto di osservazione in cui viene osservata, quest'ultima riflette la luce con egual intensità. Empiricamente questa situazione è tanto più verificata quanto più le superfici presentano geometrie non troppo complesse, quindi poco spigolose, e caratterizzate da materiali altamente riflettenti. Una palla da biliardo può essere facilmente approssimata ad un oggetto *Lambertiano*. Purtroppo tale proprietà spesso non viene mai rispettata, in quanto gli oggetti che si vogliono ricostruire il più delle volte sono geometricamente complessi e caratterizzati da superfici ruvide. Un'altra caratteristica trascurabile, è il contesto in cui sono situati i nostri oggetti, infatti il background deve essere il più uniforme possibile, l'ideale sarebbe che fosse uno sfondo nero tale da evidenziare le caratteristiche geometriche dell'oggetto. Non sempre però tutte queste specifiche possono essere soddisfatte, anzi nella realtà non lo sono quasi mai. Ragionando sempre in ambito archeologico e al ritrovamento fossile, quest'ultimo difficilmente sarà regolare e liscio ma piuttosto presenterà grosse deformazioni e irregolarità dovute all'erosione del tempo, per non parlare del background che difficilmente sarà uniforme e privo di asperità. Tutti questi fenomeni di non idealità si riflettono sulla propagazione degli errori. Poiché comportano come input, dati piuttosto perturbati, per cui nostri output saranno di conseguenza affetti da una grossa componente di rumore. Possono però essere adottate delle precauzioni per limitare il mal condizionamento dei nostri dati. Una di queste sicuramente consta nello scegliere il numero corretto di immagini. Infatti maggiore sarà il numero dei frame minore sarà il rumore presente in uscita, ma non a caso precedentemente abbiamo utilizzato l'espressione :"*numero corretto*". Infatti al crescere della quantità di immagini aumenta anche il carico computazionale. In altri termini dobbiamo trovare un giusto compromesso tra la fedeltà dei risultati e i tempi di calcolo. Ulteriori misure possono essere prese in fase preprocessing, effettuando filtraggi immagine e applicando algoritmi di edge-detection per determinare i contorni dell'oggetto ed eliminare tutto il superfluo che non è di interesse. Ciò consentirebbe di eliminare le possibili asperità presenti sullo sfondo. Questi piccoli accorgimenti sono già sufficienti per garantire buoni risultati. In questo lavoro non ci preoccuperemo però più di tanto di tali fenomeni. Infatti faremo uso di data set creati ad hoc poiché il nostro obbiettivo consiste nell'ottimizzazione e nella sintesi degli algoritmi di ricostruzione. (Rimandiamo pertanto a testi più specifici tutti coloro che sono interessati ad approfondire l'argomento).

## 1.5 Struttura dell'algoritmo di ricostruzione

Un algoritmo di ricostruzione di superfici 3-D che fa uso della Photometric Stereo, tipicamente è suddiviso in due parti. Nella prima si estraggono le informazioni fotometriche dell'oggetto, ovvero la mappa dei gradienti e la mappa delle normali, che come detto precedentemente, ci consentono di determinare punto per punto l'orientazione del corpo solido. Mentre nella seconda, a partire dalla conoscenza dei dati fotometrici, si ricostruisce la superficie 3-D risolvendo tipicamente un problema di integrazione.

La prima fase dell'algoritmo, ovvero quella che si occupa della determinazione dei gradienti, è una procedura abbastanza standard che può essere affrontata in due modi differenti, infatti si è soliti parlare di problemi fotometrici *constrained o unconstrained*. Nel primo si conoscono tutte le diverse posizioni (in termini di coordinate x-y-z) della sorgente luminosa, mentre nel secondo tale informazione non è nota e viene recuperata a partire da una decomposizione ai valori singolari della matrice delle immagini *M*. L'oggetto del nostro lavoro e quindi quello di illustrare entrambi i metodi e confrontarli cercando il più possibile di evidenziare vantaggi e svantaggi di ciascuno. Al contrario per quanto riguarda la seconda fase dell'algoritmo (ovvero la ricostruzione vera e propria della superficie) esistono una miriade di possibilità. Non sarà nostro compito analizzarle tutte, ma piuttosto ci concentreremo su una modalità che fa uso dell'equazione di Poisson che proveremo dapprima a sintetizzare e poi ottimizzare. In realtà come si può intuire dal paragrafo precedente, un algoritmo vero e proprio di ricostruzione non presenta solo questi due macro-moduli, ma anche tutta una serie di sotto-moduli, atti all'ottimizzazione e al controllo del condizionamento dei dati in ingresso su ogni macro-modulo. Ad esempio filtraggio immagini e pulitura del back-ground. Di seguito riportiamo un'illustrazione grafica dell'algoritmo completo.

# 2 Determinazione dei gradienti

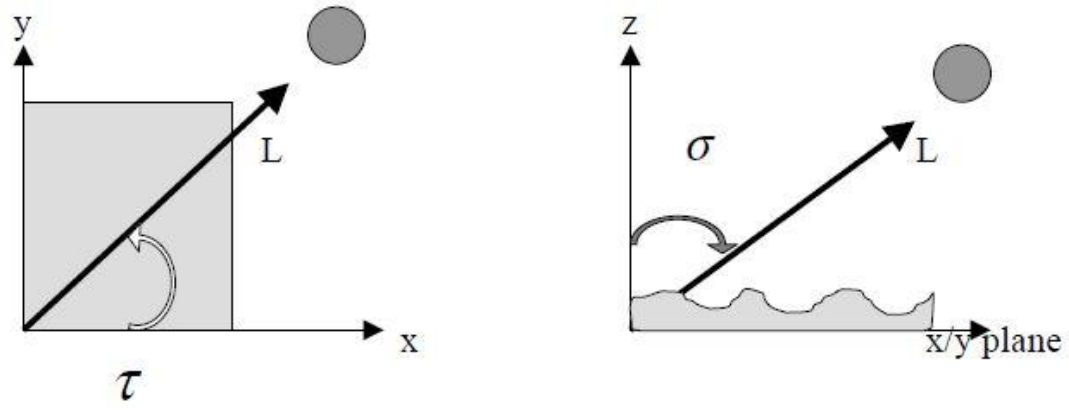
## 2.1 Introduzione

Obbiettivo di questo capitolo è mostrare ed analizzare alcuni metodi per ricavare la mappa dei gradienti, che utilizzeremo nel terzo capitolo di questo lavoro per la ricostruzione vera e propria della superficie. Come annunciato nel paragrafo 1.5, faremo riferimento a due problemi differenti, ovvero parleremo di ricerca dei gradienti: constrained, unconstrained. Nella prima fase ci limiteremo a considerare la procedura constrained, mentre nella seconda ci concentreremo su quella unconstrained, nell'ultima fase inoltre cercheremo di confrontare le due modalità in termini di prestazioni e qualità dei risultati.

## 2.2 Costruzione della mappa gradienti con approccio: "constrained"

Nella ricostruzione della mappa dei gradienti, mediante approccio constrained, è di vitale importanza conoscere le diverse direzioni luminose utilizzate durante la fase di acquisizione del data set. In altri termini ci occorre costruire una matrice $L$ di dimensione $n \times 3$, dove n indica il numero di immagini acquisite, mentre 3 è il numero che corrisponde alle coordinate x-y-z del piano di riferimento che stiamo considerando. In generale possiamo esprimere, per ogni riga della matrice $L$, una relazione del tipo:

$$l_i = \left[l_{ix}, l_{iy}, l_{iz}\right] = [\sin(\tau)\cos(\sigma), \sin(\tau)\sin(\sigma), \cos(\tau)] \qquad (2.1)$$




dove nella (2.1) $\tau$ e $\sigma$, rappresentano rispettivamente l'angolo equatoriale e azimutale in riferimento a un sistema sferico, in cui l'origine coincide con la posizione dell'oggetto che stiamo acquisendo, di conseguenza le coordinate saranno determinate a partire dalle relazioni inverse di conversione dal sistema di riferimento sferico a rettangolare. Pertanto per ogni singolo scatto, dovremmo determinare gli angoli e utilizzare la (2.1) per determinare le componenti direzionali del fascio luminoso. Tale procedura può essere svolta in due modalità distinte. In un primo caso si utilizza un approccio pratico, ovvero fotogramma per fotogramma si posiziona la sorgente luminosa in un punto e mediante della strumentazione laser si ricavano quindi gli angoli di interesse. Questa

procedura però presuppone l'utilizzo di macchinari costosi che non sempre possono essere reperiti facilmente. Oppure si può pensare di ricavare gli angoli,utilizzando una strategia che tiene conto dell'intensità associata a ogni singolo pixel delle immagini. Ad esempio si faccia riferimento alla seguente illustrazione, che raffigura un data set in cui l'oggetto acquisito è una palla da pallavolo:


è evidente come ogni zona luce risulti ben definita e marcata, si può pensare dunque di utilizzare questa informazione per ricavare gli angoli sigma e tau direttamente dai singoli frame. Per ogni immagine pertanto si individua la zona luce, e questo può essere fatto semplicemente individuando i pixel che hanno intensità massima, definendo dunque un area che se rapportata all''intera area dell'oggetto ci consente di ricavare il punto centrale della zona luce, ovvero lo *spot-light* che avrà dunque coordinata $P_x\ P_y$. Per ricavare le componenti della (2.1) possiamo a questo punto utilizzare la relazione:

$$l_i = 2(N \cdot R) \cdot N - R \qquad (2.2)$$

esprimendo N (la normale) in termini di componenti:

$$N_x = P_{x\_}C_x$$

$$N_y = P_{y\_}C_y$$

$$N_z = \sqrt{|R|^2 - {N_x}^2 - {N_y}^2}$$

dove i termini $C_x\ C_y$, indicano in termini di coordinate, il centro dell'oggetto, mentre R è un vettore che identifica la direzione di riflessione, e quest'ultima è in riferimento al punto di osservazione ovvero il luogo in cui è situata la fotocamera. Utilizzando dunque la (2.2) per ogni immagine è possibile ricavare $L$. In realtà questa operazione non è sempre possibile in accordo con quanto trattato nel paragrafo 1.4, infatti l'oggetto dovrebbe essere conforme alle ipotesi di Lambertianità, ovvero superficie liscia e riflettente in modo tale che la zona luce sia chiaramente definita. Purtroppo tali specifiche, nella maggior parte dei casi, non sono mai garantite e anche se lo fossero parzialmente porterebbero a dei risultati affetti da una grossa incertezza, motivo per il quale il primo approccio talvolta è preferibile a quello appena enunciato. Di seguito è riportata un ulteriore illustrazione in cui è raffigurato un data set non conforme alle ipotesi di Lambertianità.

Una volta chiarite dunque le diverse possibilità per ricavare la matrice $L$, possiamo fare un ulteriore passo in avanti e passare alla definizione dei gradienti. Per fare questo possiamo assumere che la superficie dell'oggetto da ricostruire sia una funzione in due variabili che possiamo vedere come una funzione del tipo: $z = f(x, y)$. Pertanto possiamo definire le componenti dei gradienti come:

$$p = \frac{\partial z}{\partial x} \qquad q = \frac{\partial z}{\partial y} \tag{2.3}$$

ovvero rappresentano le derivate direzionali della superficie nelle direzioni canoniche (1,0) e (0,1) nel punto x,y. Inoltre per ogni coppia di punti possiamo scrivere sempre due vettori tangenti perpendicolari tra loro definiti come:

$$t1 = [1,0,p] \qquad t2 = [0,1,q] \tag{2.4}$$

e calcolando il prodotto vettoriale $t1 \times t2$ si ottiene:

$$N = [-p, -q, 1] \tag{2.5}$$

che rappresenta la normale situata nel punto della superficie in cui sono stati determinati i gradienti. Dal momento che siamo interessati alla sola direzione conviene riscrivere la (2.5) effettuando un'operazione di normalizzazione:

$$N = \frac{[-p, -q, 1]}{\sqrt{1 + p^2 + q^2}} \tag{2.6}$$

Resta però ancora da capire come determinare i $p$ e $q$. Infatti questi ultimi sono stati ricavati a partire da $z$, ovvero la superficie da ricostruire, che però è la vera incognita del nostro problema. Ed è a questo punto che interviene una relazione fotometrica che ci consente di legare le componenti della matrice $L$, definita precedentemente, con la normale $N$ punto per punto:

$$I = R(p, q) = L.N \tag{2.7}$$

dove nella (2.7) $I$ è un vettore di dimensione $nx1$, in cui $n$ rappresenta il numero delle immagini, e le componenti sono note e corrispondono all'intensità luminosa associata a ogni singolo pixel. Mentre R rappresenta la riflettanza associata al medesimo pixel, che è definita come il prodotto matrice per vettore tra la matrice delle direzioni luminose e la direzione della normale nel punto i-esimo. In realtà la (2.7), è una relazione ideale, ovvero vale solo ed esclusivamente per gli oggetti che rispettano le ipotesi di Lambertianità. Ma si può estendere anche in tutti quei casi in cui le ipotesi non sono verificate:

$$I = \rho R(p, q) = \rho(L.N) \tag{2.8}$$

in questo caso $\rho$, è uno scalare, e rappresenta l'albedo. In altri termini si tratta del coefficiente di riflessione che è definito come:

$$albedo = \frac{|luce\ riflessa|}{|luce\ diretta|}$$

O alternativamente, è definito come il prodotto scalare tra normale dell'oggetto in un dato punto e la direzione del raggio luminoso che incide nel medesimo. Nel caso di un oggetto Lambertiano, tale coefficiente è costante su tutta la sua superficie ed assume sempre lo stesso valore, mentre nel caso di un oggetto non Lambertiano, quest'ultimo può assumere un valore diverso per ogni punto dell'immagine, è importante dunque tenerne conto ai fini di una corretta determinazione dei gradienti.

Se osserviamo attentamente la (2.7) e la (2.8) è immediato constatare che ci si ritrova davanti a dei sistemi lineari, è evidente dunque che per essere risolti, in accordo con la dimensione interna del prodotto matrice vettore il numero minimo di immagini necessarie deve essere almeno pari a 3. Ipotizzando dunque di avere un data set composto da sole 3 immagini, per ogni punto possiamo scrivere un vettore colonna $I$. Supponiamo inoltre di conoscere la matrice $L$ che deve essere di dimensione $3 \times 3$, sviluppando la (2.8) si ottiene:

$$\begin{bmatrix} i_{i1} \\ i_{i2} \\ i_{i3} \end{bmatrix} = \rho_i \begin{bmatrix} l_{1x} & l_{1y} & l_{1z} \\ l_{2x} & l_{2y} & l_{2z} \\ l_{3x} & l_{3y} & l_{3z} \end{bmatrix} \begin{bmatrix} n_{ix} \\ n_{iy} \\ n_{iz} \end{bmatrix} \qquad (2.9)$$

risolvendo quindi in funzione di $N$:

$$L^{-1}I = \rho.N = T \qquad (2.10)$$

dove nella (2.10) $T$ è un vettore colonna che rappresenta la soluzione di (2.10) e in accordo con (2.9) è costituito da 3 componenti. Per sostituzione diretta con la (2.6) possiamo dunque finalmente ricavare i $p, q, \rho$ come:

$$p_i = -\frac{t_1}{t_3}\ ,\ \ q_i = -\frac{t_2}{t_3}\ ,\qquad \rho_i = \sqrt{{t_1}^2 + {t_2}^2 + {t_3}^2} \qquad (2.11)$$

Ovviamente se volessimo ricostruire le mappe dei gradienti e dell'albedo complete, tale operazione dovrà essere ripetuta un numero di volte che è pari al numero di pixel che costituiscono le immagini. Ad esempio se avessimo delle immagini di dimensione 100x100, dovremmo risolvere (2.9) per ben 10000 volte. Facendo un piccolo passo indietro, come detto precedentemente per risolvere (2.9) occorrono quanto meno 3 immagini, ma questa è solo una restrizione matematica. In realtà se si vogliono ricavare i gradienti con una certa accuratezza, il numero delle immagini, deve essere decisamente superiore. Così facendo si eliminano eventuali ambiguità e si rende sovra-determinato il problema. Di conseguenza (2.9) può essere riscritto come:

$$\begin{bmatrix} i_{i1} \\ i_{i2} \\ i_{i3} \\ \vdots \\ i_{in} \end{bmatrix} = \rho_i \begin{bmatrix} l_{1x} & l_{1y} & l_{1z} \\ l_{2x} & l_{2y} & l_{2z} \\ l_{3x} & l_{3y} & l_{3z} \\ \vdots & \vdots & \vdots \\ l_{nx} & l_{ny} & l_{nz} \end{bmatrix} \begin{bmatrix} n_{ix} \\ n_{iy} \\ n_{iz} \end{bmatrix} \qquad (2.12)$$

può quindi essere risolto in termini di minimi quadrati:

$$(L^T L)^{-1} L^T I = \rho . N = T \qquad (2.13)$$

e analogamente si riutilizzano le (2.11) per ricavare i parametri di interesse.

Prima di passare all'implementazione dell'algoritmo, occorre spendere ancora due parole sull'albedo. Precedentemente ci siamo limitati a definirlo come il rapporto tra la luce riflessa e quella diretta, in realtà è necessario estendere questa definizione ai tre colori fondamentali (red,green,blue).Tutto ciò in accordo al come vengono memorizzate le immagini a colori, che possono essere viste come delle matrici tridimensionali mxnx3, in cui a ogni livello è associato il contenuto informativo nelle singole componenti luminose RGB. Quindi più correttamente l'albedo deve essere scritto, non come uno scalare, ma un vettore che tiene conto rispettivamente dei coefficienti di riflessione associati a ogni singolo canale luce: $(\rho_R, \rho_G, \rho_B)$. Quindi (2.13) in definitiva diventa:

$$\begin{cases} (L^T L)^{-1} L^T I = \rho_R . N \\ (L^T L)^{-1} L^T I = \rho_G . N \\ (L^T L)^{-1} L^T I = \rho_B . N \end{cases}$$

In realtà tale operazione è necessaria solo ed esclusivamente se si ha l'intenzione di ricostruire la superficie a colori e questa operazione non è sempre necessaria. Motivo per il quale ci limiteremo a ricostruire le superfici degli oggetti in bianco e nero, e quindi in accordo con la (2.13) potremo sempre considerare l'albedo come uno scalare.

## 2.3 Implementazione dell'algoritmo di ricostruzione dei gradienti

In questo paragrafo sintetizzeremo l'algoritmo di ricostruzione delle mappe dei gradienti descritto nel paragrafo precedente. Gli ingressi che ci occorrono sono: la matrice $L$ e la matrice delle immagini $M$. Quest'ultima è costruita a partire dalle singole immagini che costituiscono i data set che considereremo, pertanto possiamo vederla come una matrice tridimensionale di dimensione $row \times col \times n$, dove row, col e n rappresentano rispettivamente: il numero di pixel per riga, il numero di pixel per colonna e il numero delle immagini. In realtà esiste un ulteriore modo per definire $M$, infatti piuttosto che definire una matrice tridimensionale, si può pensare di costruire una matrice $n \times row * col$. Ad ogni riga i-esima corrisponde dunque un'immagine del data set che

è stata opportunamente trasformata in un vettore colonna effettuando un *reshape* $I_i(:)$. Tale processo di ottimizzazione è comunemente noto in matematica come *"ordinamento Lessigografico" e* ci consente di trarre grossi vantaggi dal punto di vista computazionale. Infatti piuttosto che gestire tre cicli annidati, è sufficiente gestirne solamente uno. Questa operazione risulta pertanto di grande importanza, soprattutto in quei casi in cui si ha a che fare con linguaggi di programmazione vettoriale (Octave,Matlab). Di seguito riportiamo la procedura adottata per la costruzione di M:

```
data_set(picture_1, picture_2,picture_3,....,picture_n),
n=size(data_set),
[row,col]=size(data_set(1)),
for i=1:l
M(i,1)=reshape(data_set(i),1,row*col),
end
```

Le diverse immagini ovviamente hanno tutte la stessa dimensione e coerentemente a quanto detto nella fine del paragrafo precedente sono in bianco e nero. A questo punto una volta definiti gli ingressi possiamo passare alla stesura dello pseudo codice dell'algoritmo:

```
Input(M, L,row,col)
G=(L^T L)^-1 L^T,

for i=1:row*col
I=M(:,i),
T=G*I,

albedo(i)=norm(T),
p(i)= -T(1,1)/T(3,1),
q(i)= -T(21)/T(3,1),

end

albedo=reshape(albedo,row,col),
p=reshape(p,row,col),
q=reshape(q,row,col),

Output(albedo,p,q)
```

Come prima operazione calcoliamo la matrice $G$ facendo riferimento all'espressione (2.13), tale scelta è conseguente al fatto che $L$ è sempre la stessa e di conseguenza $G$. Successivamente ha inizio la fase iterativa, a ogni iterazione salviamo su una variabile d'appoggio $I$ che è un vettore colonna di dimensione pari a $n$ (si veda paragrafo 2.3), quindi il vettore colonna soluzione $T$ sarà dato dal risultato del prodotto matrice per vettore enunciato in (2.13). Infine calcoliamo utilizzando le (2.11) i gradienti e l'albedo, questa operazione sarà dunque ripetuta per un numero di volte pari a $row \times col$. In conclusione si ottengono dunque dei vettori riga di dimensione $(1, row \times col)$ che

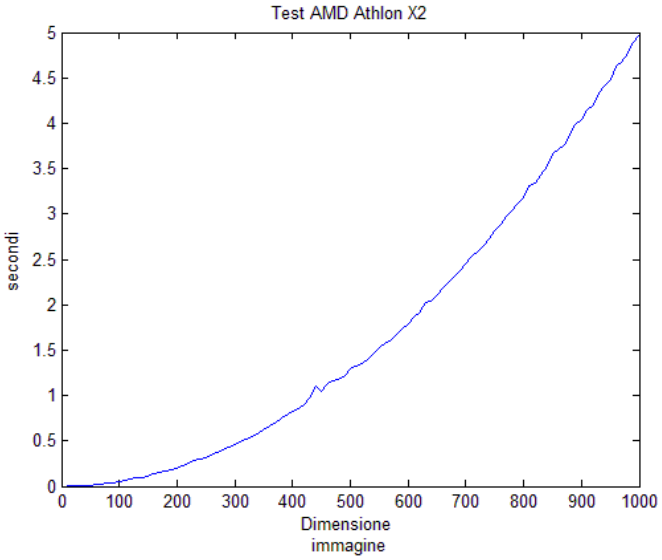
dobbiamo quindi riportare in forma matriciale semplicemente effettuando un reshape finale, tali matrici costituiranno quindi i nostri output.

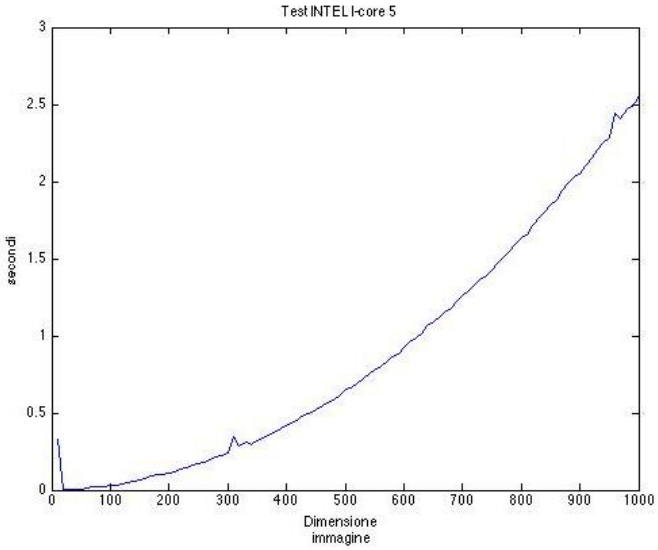
In termini computazionali ciò che incide nell'algoritmo è la dimensione delle immagini, la scelta però come prima cosa di costruirci $G$, ci consente piuttosto che risolvere un sistema a ogni iterazione, di effettuare dei prodotti matrice per vettore che possiamo calcolare in modo piuttosto rapido al posto di utilizzare metodi di risoluzione diretti. Paghiamo dunque $\boldsymbol{O}(n^3)$ per l'inversione iniziale di $(L^T L)^{-1}$ ma si tratta comunque di una matrice 3x3, il resto è dato sostanzialmente da prodotti. Di seguito riportiamo inoltre l'implementazione dell'algoritmo utilizzando MATLAB:

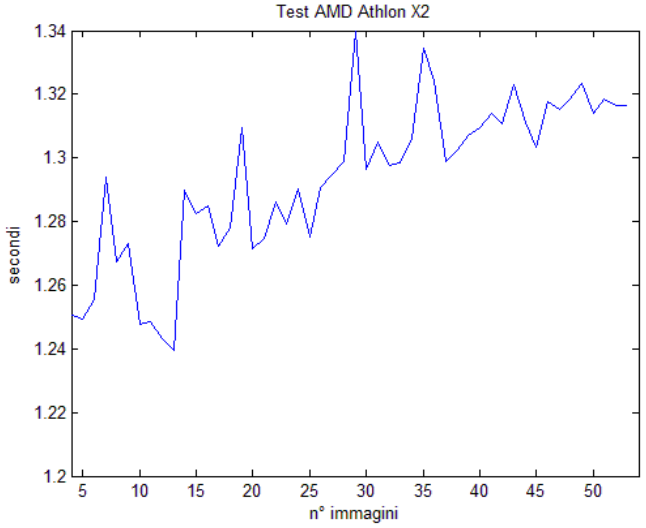
```
function[p q N albedo]=fotometric(M,L,row,col,gain)



albedo=zeros(row,col),
p=zeros(row,col),
q=zeros(row,col),



G=(inv(L'*L))*L',

for i=1:row*col
   int=M(:,i),
   T=(G *int),

   albedo(i)=norm(T),

   p(i)=(-T(1,1)/T(3,1)),
   q(i)=(-T(2,1)/T(3,1)),

end



%ridimensionamento dei gradienti
albedo=reshape(albedo,row,col)*gain,
p=reshape(p,row,col),
q=reshape(q,row,col),


%calcolo della normale
N=albedo.*(p./sqrt(1 + p.^2 + q.^2)),
N(:,:,2)=albedo.*(q./sqrt(1 + p.^2 + q.^2)),
N(:,:,3)=albedo.*(1./sqrt(1 + p.^2 + q.^2)),

%raffinamento
p(isnan(p))=0,
q(isnan(q))=0,
N(isnan(v))=0,
albedo(isnan(albedo))=0,
p(isinf(p))=0,
q(isinf(q))=0,
N(isinf(v))=0,
albedo(isinf(albedo))=0,
```

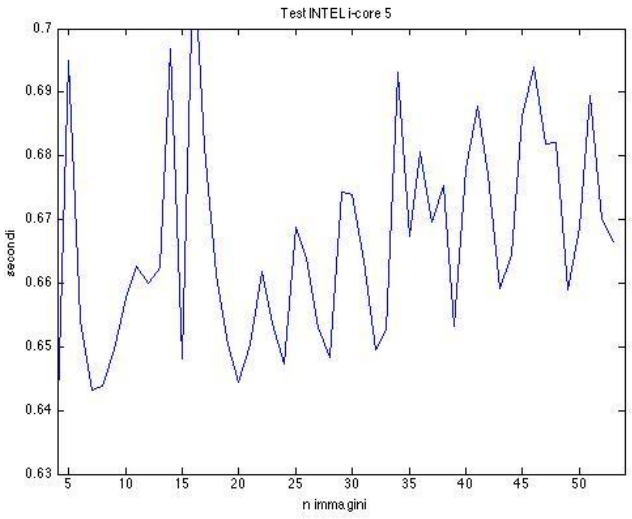
```
p=p.*albedo,
q=q.*albedo,
end
```

Differentemente a quanto illustrato nello pseudo codice, abbiamo ricavato anche la mappa delle normali $N$ che abbiamo implementato facendo uso della (2.6), inoltre nella parte conclusiva (raffinamento) ci siamo preoccupati di effettuare un operazione di filtraggio di tutti gli elementi NaN(non definiti) e dei termini -inf che possono determinarsi a partire dai processi di arrotondamento macchina. Tale procedura per quanto non abbia alcun effetto sulla visualizzazione stessa degli elementi, consente però una migliore e accurata ricostruzione della superficie. Infine i gradienti punto per punto vengono moltiplicati per il corrispettivo valore dell'albedo. Di seguito sono riportati inoltre dei grafici per valutare le prestazioni dell'algoritmo:



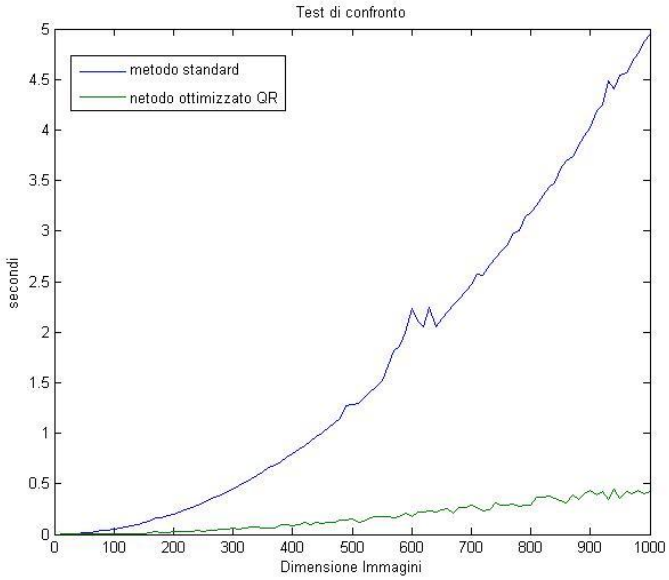
I test sono stati effettuati secondo 2 modalità differenti: nel primo abbiamo tenuto fisso il numero delle immagini, pari a 5, facendo variare solamente la dimensione dei frame per valori compresi da 10x10 a 1000x1000, nel secondo invece il viceversa, quindi tenendo fissa la dimensione delle immagini a 500x500 abbiamo fatto variare il numero da 3 a 54. Non solo ma tali esperimenti sono stati ripetuti utilizzando due calcolatori differenti, in un caso utilizzando un architettura AMD dual core a 2.8 GHz , e nell'altro una INTEL i-core 5 sempre da 2.8 Ghz. Come è constatabile dai risultati l'andamento delle curve e pressoché identico in entrambe le architetture, ad eccezione ovviamente della scala dei tempi, che nel caso INTEL risulta dimezzata rispetto ad AMD. Interessante inoltre osservare come l'andamento del primo test assumi un andamento parabolico al crescere della dimensione delle immagini. Ciò è abbastanza ovvio dal momento che abbiamo a che fare con prodotti, pertanto la complessità è pari a $\boldsymbol{O}(n^2)$ il che giustifica l'andamento. Mentre nel secondo grafico, come si può constatare dalla scala dei tempi, l'andamento è meno definito e oscillante, ma è compreso in un range di valori molto piccolo. Pertanto si può concludere, in accordo con quanto detto pocanzi, che il carico computazionale non dipende tanto dal numero delle immagini, ma dalla loro dimensione. Questo è un risultato abbastanza incoraggiante in virtù del fatto che maggiore è il numero delle immagini, maggiore sarà la qualità dei risultati, pertanto a fronte di tali considerazioni, un algoritmo di questo tipo si presta molto bene in tutti quei casi in cui le immagini siano di dimensione medio grande senza preoccuparsi tanto del numero di quest'ultime,che può essere anche elevato, a patto però di costruire con precisione la matrice $L$ e quindi di conoscere con accuratezza la misura degli angoli sigma e tau fotogramma per fotogramma.

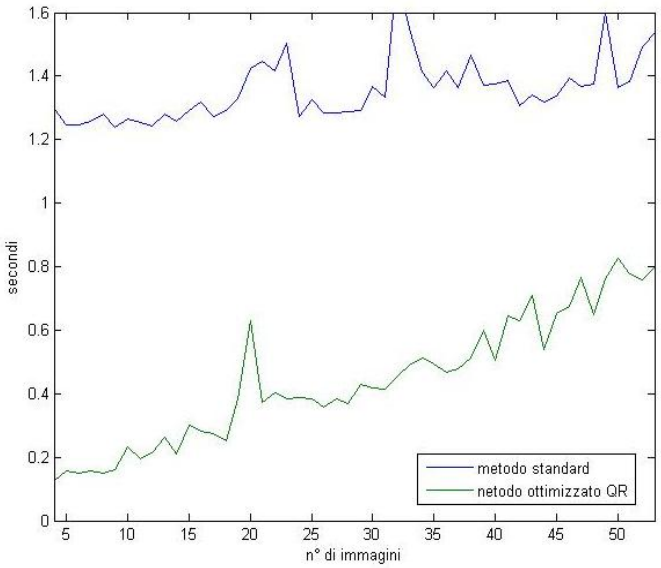
### 2.3.2 Ottimizzazione

Sembrerebbe dunque che l'algoritmo appena presentato sia abbastanza valido in termini di tempo di calcolo, soprattutto se si confronta con alcuni risultati riportati in letteratura, questo sopratutto deriva dalla scelta di aver costruito $M$ secondo l'*ordine lessigografico*. In realtà sfruttando l'algebra lineare si può fare addirittura meglio. Dal momento che $L$ è sempre la stessa, si può pensare di costruire un'opportuna fattorizzazione QR per la matrice delle direzioni di luce e quindi possiamo ottenere $T$ come:

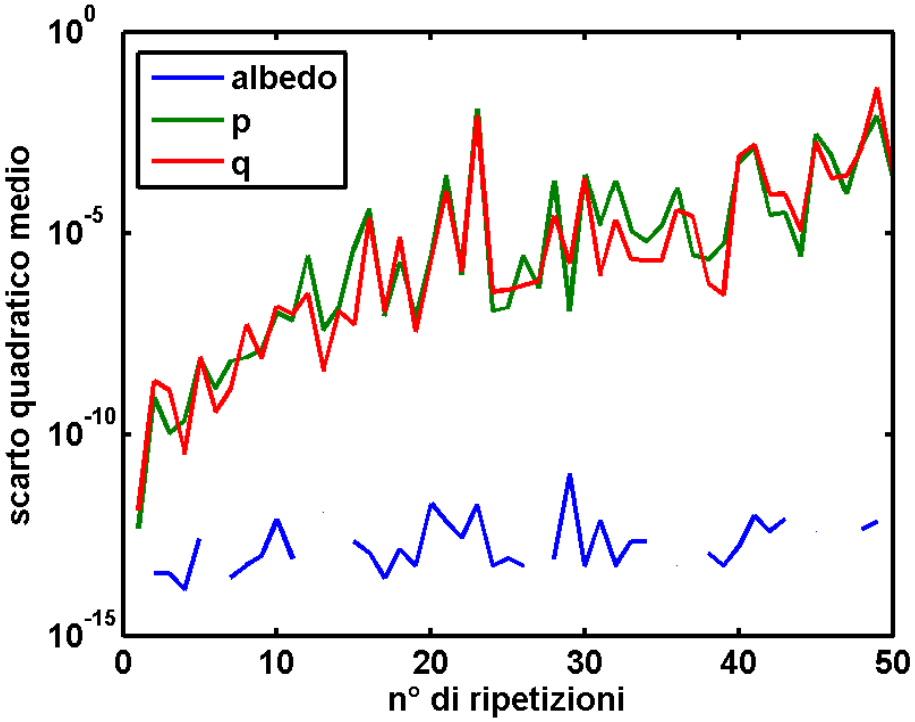
$$T = R^{-1}(M^T Q)^T \qquad (2.14)$$

dove $T$ a questo punto è una matrice di dimensione $3 \times row * col$, da cui possiamo ricavare direttamente i gradienti e l'albedo utilizzando le (2.3) punto per punto ed effettuare quindi un reshape finale per riportarci a matrici di dimensione $row \times col$ . Così facendo, piuttosto che ricavarci i termini $t_i$ iterativamente, li otteniamo tutti in un unico passaggio. In questo caso non riportiamo lo pseudo codice e l'implementazione eseguita in Matlab, perché di fatto l'unica differenza rispetto ai codici precedenti, risiede nell'aver applicato direttamente la (2.14) mentre tutto il resto rimane inalterato. Piuttosto siamo interessati a spendere qualche ulteriore parola e mostrare qualche risultato in merito al confronto diretto tra la versione "standard", discussa precedentemente, e quella ottimizzata affrontata in questo sotto paragrafo. Per far ciò partiamo dunque mostrando alcuni risultati in relazione ai tempi di calcolo:




I test eseguiti sono identici a quelli precedenti. Per cui nel primo caso si è fatta variare la dimensione delle immagini, mentre nel secondo si è fatto variare il loro numero. Dal momento che i test ottenuti sarebbero stati equivalenti, ci siamo limitati ad applicarli solo ed esclusivamente al calcolatore con architettura AMD. E' evidente come i risultati ottenuti siano nettamente in favore al metodo ottimizzato che consente di ridurre i tempi di calcolo, in ambo i casi, sino ad un ordine di grandezza circa. Successivamente ci siamo posti il problema di verificare che i risultati di elaborazione, in termini di gradienti e albedo fossero equivalenti al metodo standard. Per ogni iterazione abbiamo dunque calcolato lo scarto quadratico medio tra i risultati elaborati con il metodo standard e quello ottimizzato. Di seguito riportiamo il grafico dove è constatabile, a meno di qualche picco, dovuto al fatto di aver utilizzato input random, come i gradienti e l'albedo effettivamente coincidano.



## 2.4 Costruzione della mappa gradienti con approccio: "unconstrained"

A differenza di quanto trattato nel metodo di ricostruzione dei gradienti in modalità constrained, in questo paragrafo, ci concentreremo sullo sviluppo di un metodo di ricostruzione senza fare uso della matrice $L$, quindi utilizzando come ingressi solo ed esclusivamente le immagini fornite dai data set. Per tale motivazione l'approccio è anche comunemente noto in letteratura come: *unkown lights directions.* Lo sviluppo di tale metodologia, nella storia della Photometric Stereo, si colloca in un in un arco temporale compreso tra gli inizi degli anni 2000 e 2006, dove è concentrato il maggior numero di articoli scritti dai vari gruppi di ricerca [8] [9]. Da dopo questo periodo in poi, non si hanno più notizie su ulteriori studi e sviluppi. Il che ci fa pensare che parte delle evoluzioni siano state volutamente celate oppure si sia arrivato a un punto di svolta tale da portare ad abbandonare qualsiasi studio intrapreso (il punto di svolta è a noi ignoto). Nonostante questo buio profondo, i diversi articoli analizzati ci hanno interessato particolarmente, motivo per il quale abbiamo voluto provare a sintetizzare un algoritmo di elaborazione che ci ha particolarmente sorpreso. Non solo ma l'idea di non doversi preoccupare ad ogni acquisizione di misurare con accuratezza la direzione della luce, ci pare un grosso vantaggio.

Nel capitolo precedente abbiamo visto come fosse possibile estrarre i gradienti a partire dalla conoscenza della matrice delle immagini $M$ e della matrice $L$. Supponendo a questo punto di conoscere solo la matrice $M$, possiamo in realtà scrivere un integrale di convoluzione su una sfera che punto per punto ci restituisce un certo valore di riflettanza. Tale integrale ha come argomento proprio le componenti della luce in bassa frequenza e l'intensità del pixel i-esimo. L' integrale non può essere risolto nella sua forma canonica, però in realtà e possibile scomporlo in serie mediante armoniche sferiche. Se ci limitiamo a una scomposizione in serie del primo ordine quello che otteniamo è un spazio formato da quattro componenti, che nella fattispecie, corrispondono all'albedo alle tre componenti della normale dell'oggetto. In altri termini ci siamo riportati su uno spazio di tipo 4D che nell'ambito fotometrico è una rappresentazione delle prime quattro armoniche sferiche. Il fatto si aver rappresentano in termini di polinomio il nostro integrale ci consente di fatto di rendere determinato il problema e ridurre drasticamente il numero delle incognite. Alcuni risultati forniti dalla letteratura dimostrano che con un'approssimazione del primo ordine è possibile ricavare sino all'ottanta per cento delle informazioni geometriche dell'oggetto, il che nella maggior parte delle applicazioni è sicuramente più che sufficiente. In realtà altri studi si dimostra che in certi casi, soprattutto quando non si ha a che fare con superfici non Lambertiane, l'approssimazione del primo ordine non sia sufficiente. Motivo per il quale è opportuno sviluppare in serie l'integrale sino al secondo ordine ottenendo uno spazio 9D in cui all'interno sono contenute le prime nove armoniche sferiche. In realtà in questo testo noi ci limiteremo al solo caso 4D rimandando ad eventuali studi futuri l'analisi di tale ulteriore evoluzione.

Per prima cosa partiamo dunque dalla seguente relazione:

$$M = LS_4 \qquad (2.15)$$

dove $M$ è la solita matrice delle immagini, quindi di dimensione $n \times row * col$, che dall'analisi precedentemente effettuata può essere definita come: il prodotto matriciale tra la matrice delle componenti in bassa frequenza della luce, di dimensione $n \times 4$ e una matrice $S_4$ , di dimensione $4 \times row * col$, in cui sono contenute le prime quattro armoniche sferiche conseguenti alla sviluppo in serie dell'integrale di convoluzione. Il punto immediatamente successivo consta nella determinazione di $L$ ed $S$ che possono essere ricavate in varie modalità. Sicuramente la più adottata ed efficiente consiste nell'utilizzo della *principal-component- analisys* [11] che prevede l'utilizzo della fattorizzazione SVD (singular value decomposition). In altri termini possiamo riscrivere la nostra matrice delle immagini come:

$$M = U\,\Delta\,V^T \qquad (2.16)$$

dove U è una matrice unitaria ($U\,U^* = U^*U = I$ ) ortogonale se $k$ è reale, ma tipicamente nel caso di interesse lo è sempre. Δ è una matrice diagonale che contiene i valori singolari e infine V è una matrice di rotazione anch'essa unitaria. A partire dunque dalla decomposizione (2.16) possiamo ricavare $L$ ed $S$ come:

$$L = U\sqrt{\Delta} \qquad S = \sqrt{\Delta}V^T \qquad (2.17)$$

La (2.16) rappresenta una decomposizione completa della matrice delle immagini che tipicamente assume dimensioni abbastanza importanti, e quindi utilizzare la SVD sembrerebbe un rischio in termini computazionali. In accordo però con quando detto in precedenza, noi non siamo interessati allo sviluppo completo di $M$ ma alle prime sue quattro componenti armoniche. Per tale motivazione possiamo limitare l'analisi ai soli primi quattro valori singolari. Di conseguenza piuttosto che utilizzare una decomposizione completa possiamo fare uso di una decomposizione con fattore di troncamento pari a quattro. Evidentemente ciò impone che il numero minimo di immagini debba essere proprio pari a quattro, contrariamente a quanto visto nel metodo precedente in cui il numero minimo era fissato a tre.

Facendo uso di TSVD (2.15) può essere riscritta come:

$$M \approx U_4\,\Delta_4\,{V^T}_4 \qquad (2.18)$$

dove il pedice sta appunto a segnalare il fattore di troncamento utilizzato. Conseguentemente alla (2.18) possiamo ridefinire (2.17) come:

$$\tilde{L} = U_4\sqrt{\Delta_4} \qquad \tilde{S} = \sqrt{\Delta_4}{V^T}_4 \qquad (2.19)$$

dove $\tilde{L}$ è una matrice di dimensione pari a $n \times 4$ ed $\tilde{S}$ è una matrice di dimensione $4 \times row * col$. Il fatto di aver decomposto $M$ con un fattore di troncamento ci ha sicuramente consentito di ridurre il carico computazionale ma pone altresì un problema di ambiguità. Infatti le (2.19) differiscono quantitativamente dalle (2.16) e per poter essere utilizzate è necessario eseguire un ulteriore passaggio. Possiamo dunque domandarci se esiste una qualche trasformazione lineare , meglio nota come trasformazione di *Lorentz,* tale che:

$$M = \tilde{L}\, A^{-1} A\ \tilde{S} = L\, S \qquad (2.20)$$

Dove A rappresenta la matrice di trasformazione che coerentemente con le (2.15) e (2.17) deve essere di dimensione $4 \times 4$. Per ricavare $A$ possiamo partire da una considerazione fisica, infatti ogni riga di $S(\ \vec{s} = (s_1, s_2, s_3, s_4)^T)$ soddisfa la seguente condizione:

$$s_1{}^2 = s_2{}^2 + s_3{}^2 + s_4{}^2 \qquad (2.21)$$

La (2.21) è sempre valida in tutti quei casi in cui $s_1$ rappresenta l'albedo in uno specifico punto ed $s_2, s_3, s_4$ solo le componenti della normale nel medesimo. Tale vincolo può essere espresso in termini matriciali come:

$$\vec{s}^T J \vec{s} = 0 \qquad (2.22)$$

dove $J$ è una matrice diagonale definita come: $J = diag\{-1,1,1,1\}$. In conformità con quanto detto precedentemente, la (2.22) non vale per $\tilde{S}$ . Ma sfruttando la (2.20) possiamo in realtà scrivere per ogni generica riga di $S$ una relazione del tipo:

$$\vec{s} = A\, \vec{q} \qquad (2.23)$$

dove $\vec{q}$ rappresenta una generica riga, tra le 4, della matrice $\tilde{S}$ (si faccia attenzione a non confondere il $\vec{q}$ con la definizione di gradiente data nel paragrafo 2.2). Sostituendo a questo punto la (2.23) nella (2.22) si ottiene una trasformazione del tipo:

$$\tilde{\vec{q}}^T A^T\, J\, A\, \vec{q} = 0 \qquad (2.24)$$

in cui possiamo definire un ulteriore matrice $B = A^T\, J\, A$ che evidentemente ha dimensione $4 \times 4$, dato che $J$ ed $A$ sono al loro volta di diemansione $4 \times 4$. In conclusione possiamo riscrivere quindi (2.24) come:

$$\tilde{\vec{q}}^T B\, \vec{q} = 0 \qquad (2.25)$$

Questa equazione è lineare e omogenea e vorremmo utilizzarla per ricavare le componenti della matrice $B$(e di conseguenza $A$), che per come è stata definita è una matrice simmetrica e quindi il numero delle componenti da determinarsi è pari a dieci, in quanto una volta nota la diagonale e i termini del triangolo superiore, per simmetria è noto anche il triangolo inferiore. Di conseguenza abbiamo la necessità di determinare un vettore soluzione così fatto:

$$b_{m,n} = (b_{1,1}, b_{2,2}, b_{33}, b_{44}, b_{1,2}, b_{1,3}, b_{1,4}, b_{23}, b_{24}, b_{34})$$

quest'ultimo può essere ricavato a partire dalla conoscenza della matrice $\tilde{S}$. Per ogni punto dunque possiamo scrivere un'equazione del tipo:

$$b_{1,1} q_1{}^2 + b_{2,2} q_2{}^2 + b_{33} q_3{}^2 + b_{44} q_4{}^2 + b_{1,2} q_1 q_2 + b_{1,3} q_1 q_3 +$$

$$b_{1,4} q_1 q_4 + b_{23} q_2 q_3 + b_{24} q_2 q_4 + b_{34} q_3 q_4 = 0$$

che può essere scritta in una forma più compatta come:

$$Q\ b = 0 \qquad (2.26)$$

dove $Q$ è una matrice di dimensione $row * col \times 10$ che è costruita a partire dalla conoscenza di $\tilde{S}$. In altri termini per ogni singola riga $q$ di $\tilde{S}$, le 10 colonne di $Q$ sono definite in accordo con il vettore soluzione, come: $({q_{1,1}}^2, \dots, {q_{4,4}}^2, 2q_1q_2, \dots, 2q_3q_4)$. Pertanto possiamo trovare le componenti di $B$, a meno di un fattore di scala, osservando l'ampiezza dello *spazio nullo* di ogni singola colonna di $Q$. In realtà dal momento che le colonne di Q sono tutte linearmente indipendenti è più corretto parlare di *spazio approssimato nullo,* e per fare ciò si può pensare nuovamente di ricorrere all'utilizzo della decomposizione per valori singolari.

Una volta quindi trovati i coefficienti $b_{i,j}$ , possiamo finalmente ricavare la matrice $B$, a meno di un fattore scalare tale per cui:

$$\tilde{B} = \beta\ B = \beta(A^T\ J\ A) \qquad (2.27)$$

da cui possiamo pertanto ricavare la matrice di ambiguità $A$ ,che è l'elemento che ci siamo prefissati di ricavare sin dall'inizio. Sinché $B$ è simmetrica i suoi quattro autovalori saranno tutti reali. Non solo quest'ultimi presentano una strutta di segno, infatti possono presentarsi i seguenti due casi: il primo autovalore è negativo, mentre tutti gli altri tre sono positivi, viceversa, il primo è positivo e i rimanenti sono negativi. Per spiegare questo fenomeno si può fare riferimento alla seguente relazione:

$$\det(\tilde{B}) = \det(\beta(A^T\ J\ A)\ ) =$$

$$= \beta^4 \det(J) det^2(A) = -\beta^4 det^2 A \qquad (2.28)$$

Conseguentemente, se $A$ è non singolare e $\beta$ è diverso da 0, il determinate di B sarà sempre un quantità negativa. Questo è abbastanza evidente dal momento che per definizione, il determinate è dato dal prodotto degli autovalori. Possiamo dunque sfruttare questa proprietà per decomporre $B$ nel seguente modo:

$$B = W\ J\ \Lambda\ W^T \qquad (2.29)$$

dove le quattro colonne di $W$ rappresentano gli autovettori di B, mentre la matrice $\Lambda$ contiene i suoi autovalori. Sfruttando tale decomposizione possiamo finalmente ricavare $A$ come:

$$A = \sqrt{\Lambda} W^T \qquad (2.30)$$

e quindi utilizzando le (2.20) $S$ ed $L$. La prima riga di S corrisponde all'albedo, mentre le restanti costituiscono le componenti della normale dell'oggetto che si vuole ricostruire, in altri termini:

$$\begin{cases} s_1 = \rho \\ s_2 = N_x = \dfrac{p}{\sqrt{1+p^2+q^2}} \\ s_1 = \rho \\ s_3 = N_z = \dfrac{q}{\sqrt{1+p^2+q^2}} \\ s_1 = \rho \\ s_4 = N_z = \dfrac{-1}{\sqrt{1+p^2+q^2}} \end{cases} \qquad (2.31)$$

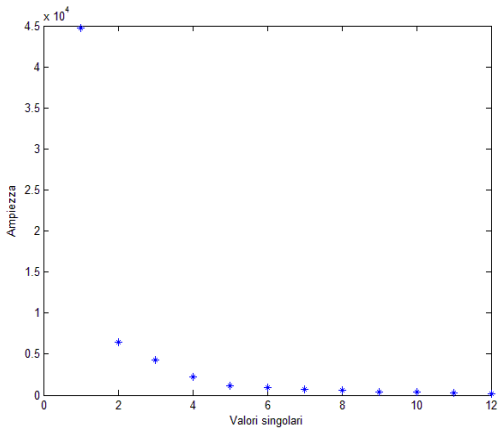
A partire da (2.31) riotteniamo dunque i gradienti definiti in 2.3 come:

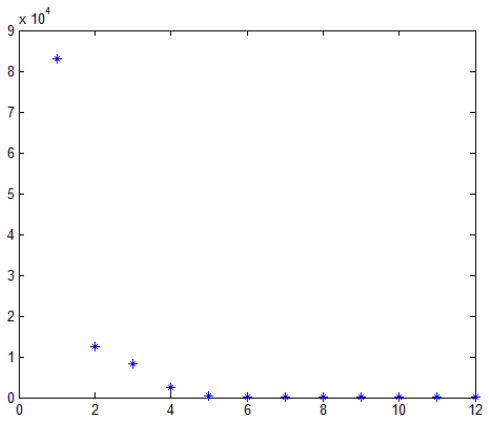
$$p = -\frac{s_2}{s_4} \qquad\qquad q = -\frac{s_3}{s_4} \qquad (2.32)$$

Per quanto riguarda la matrice delle direzioni luce, quest'ultima e di dimensione pari a $n \times 4$, contrariamente a quanto visto nei paragrafi precedenti, in cui la dimensione era $n \times 3$. In realtà la matrice è la stessa, il quattro esce fuori semplicemente perché ci siamo messi su uno spazio 4D in riferimento alle componenti delle quattro armoniche sferiche. Tutto questo in definitiva per dire che una delle 4 colonne è sempre ridondante.

Quindi riassumendo, siamo partiti con la decomposizione della matrici delle immagini utilizzando un fattore di troncamento pari a quattro (four-harmoncs-method) e ci siamo poi posti successivamente il problema di trovare una trasformazione lineare che ci consentisse di ricavare, direttamente a partire dalla decomposizione, le informazioni di interesse. Tutto questo è stato fatto senza conoscere alcunché in merito le direzioni della luce, che in realtà siamo riusciti comunque a determinare passando per altra via.

In realtà rimane un altro aspetto da verificare prima di passare all'implementazione, infatti tipicamente quando si parla di *principal component analysis* [12] deve essere giustificata la scelta di aver utilizzato come fattore di troncamento $k$ il valore 4. In altri termini dobbiamo in qualche modo dimostrare che dopo il quarto valore singolare in poi vi sia veramente un crollo repentino dell'ampiezza dei successivi termini, in accordo con quanto enunciato nel *four-harmonics method,* ovvero che il grosso contenuto informativo risiede nei primi quattro valori singolari . Per fare questo abbiamo semplicemente effettuato vari test con diversi data set di seguito proponiamo alcuni risultati:



I seguenti test sono stati effettuati utilizzando delle matrici di ingresso $M$ di dimensione $12 \times row * col$ , che sono state decomposte sino al dodicesimo valore singolare (il massimo possibile dato che M è rettangolare). Effettivamente dal quarto valore singolare in poi si registra un decadimento di circa il 70 % in entrambe i casi. Tale decadimento inoltre pare essere in funzione del numero di immagini contenute nel data set, e quindi maggiore è quest'ultimo tanto più ampio sarà il dislivello. In realtà abbiamo registrato anche altri casi in cui questo trend pare non venga rispettato. Infatti la qualità dei data set pare influire molto sull'ampiezza dello scalino. Pertanto tale indicatore in realtà può essere utilizzato ai fini di controllo della qualità del data set stesso ed è evidente che maggiore sarà il dislivello tra il quarto valore singolare e tutti gli altri, tanto maggiore sarà la qualità dei gradienti che si ottengono dall'elaborazione. Il che comporta avere ovviamente delle ricostruzioni fedeli. Esistono inoltre in letteratura altre interpretazioni in merito alla scelta del quattro come fattore di troncamento. Per esempio se si ha a che fare con un oggetto immerso in uno scenario, e tale oggetto ricopre la maggior parte dell'immagine, nei primi quattro valori singolari è contenuta la maggior parte delle informazioni geometriche della superficie dell'oggetto. Pertanto non è detto che dopo il quarto valore singolare sia d'obbligo avere un brusco decadimento, anche perché interverrebbe l'azione dello scenario che contiene dell'informazione. In altri termini scegliere come valore singolare il fattore quattro implica filtrare tutto ciò che è in più rispetto all'oggetto, a patto però che quest'ultimo ricopra una regione piuttosto ampia dello scenario in cui è immerso, tutto questo ovviamente in accordo con il discorso legato alle quattro armoniche sferiche imbastito all'inizio di questo paragrafo.

## 2.5 Implementazione dell'algoritmo di ricostruzione dei gradienti (Unknown Lighting)

In questo paragrafo ci occuperemo di implementare un algoritmo che ci consenta di ricavare i gradienti a partire dal metodo descritto nel paragrafo precedente. Differentemente con quanto visto in 2.5, qui abbiamo solo un ingresso, la matrice delle immagini $M$ che possiamo ricavare nel medesimo modo descritto in 2.5. Inoltre assumiamo che tutte le immagini presenti del data set siano in bianco e nero. Riassumendo dunque brevemente i passi principali dell'algoritmo: procediamo per prima cosa con la decomposizione della matrice $M$ utilizzando un fattore di troncamento pari a quattro. Costruiamo successivamente la matrice Q che utilizzeremo per determinare lo spazio nullo e quindi i dieci coefficienti della matrice $B$. Decomponendo quest'ultima, calcolando gli autovalori e gli autovettori, ricaviamo di conseguenza la matrice $A$. Infine a partire dalla conoscenza della matrice di ambiguità, ricaviamo quindi $S$ ed $L$. Di seguito riportiamo lo pseudo codice come traccia che utilizzeremo come riferimento per l'implementazione in Matlab:

***Input****(M,row,col)*
*K=4,*

*[U delta V]= **svd**(M,k),*

*L_=U*sqrt(delta),*
*S_=sqrt(delta)*V'*

*Q=[S_(1,:)^2 S_(2,:) S_(3,:) S_(4,:) 2*S_(1,:)*S_(2,:) 2*S_(1,:)*S_(3,:) 2*S_(1,:)*S_(4,:) 2*S_(2,:)*S_(3,:) 2*S_(2,:)*S_(4,:) 2*S_(3,:)*S_(4,:) ],*

b=***null_space***(Q),

*B=[ b(1) b(5) b(6) b(7), 0 b(2) b(8) b(9), 0 0 b(3) b(10), 0 0 0 b(4)],*

[eigenvectors eigenvalue ]=**eig**(B),

A=sqrt(abs(eigenvalue))*eignevectors',

S=A*S_,
L=A*L_,

albedo=S(1,:),
nx=S(2,:),
ny=S(3,:),
nz=S(4,:),

p=-nx./nz,
q=ny./nz,

*albedo=reshape(albedo,1,row*col),*
*p=reshape(p,1,row*col),*
*q=reshape(q,1,row*col)*

***Output**(albedo,p,q)*

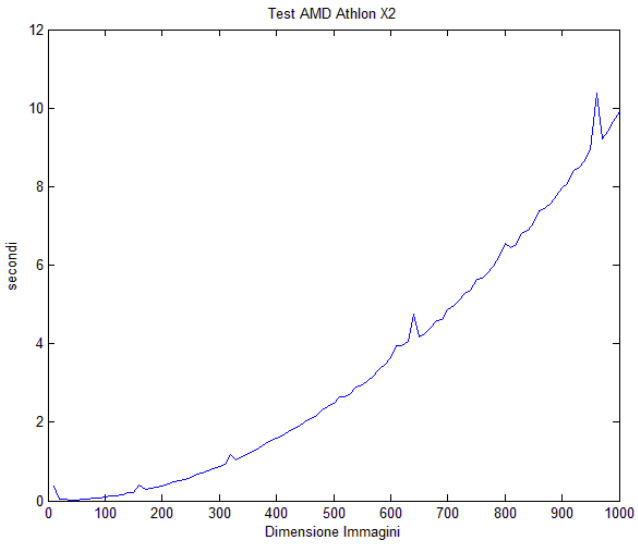
Implementazione Matlab:

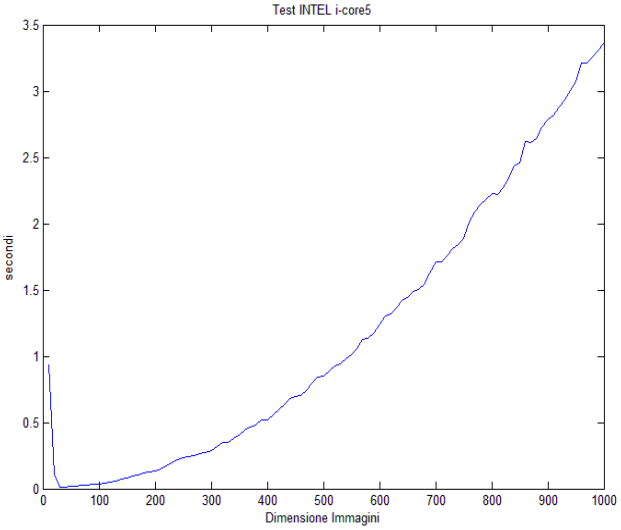
```
function[p q N albedo L2]=Q(M,row,col,gain)

[U delta V]=svds(M,4),
S=(sqrt(delta)*(V')),
L=U*(sqrt(delta)),
[m,n]=size(S),
```

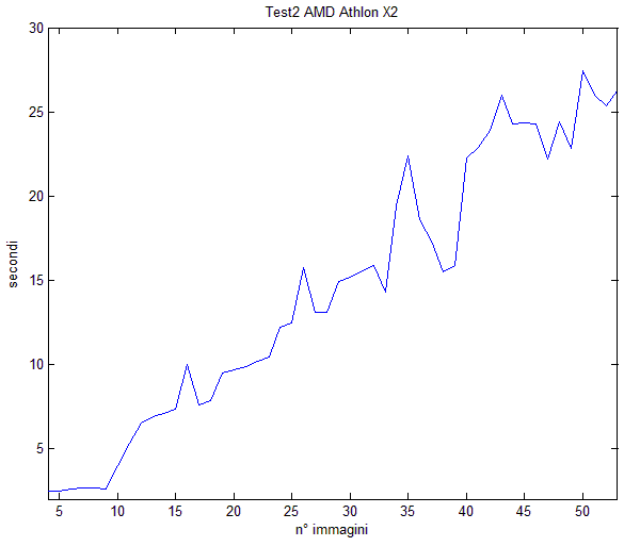
```
%costruisco la matrice Q
   q1=(S').^2,
   r1(1:1:n,1)=S(1,1:1:n),
   r2(1:1:n,1)=S(2,1:1:n),
   r3(1:1:n,1)=S(3,1:1:n),
   r4(1:1:n,1)=S(4,1:1:n),
   q2=2*r1.*r2,
   q3=2*r1.*r3,
   q4=2*r1.*r4,
   q5=2*r2.*r3,
   q6=2*r2.*r4,
   q7=2*r3.*r4,

%determinazione dello spazio nullo delle colonne di Q
Q=[q1,q2,q3,q4,q5,q6,q7],

[U,g,V2] = svd(Q,0),
z =(V2(:,end)),

B=[z(1) z(5) z(6) z(7),z(5) z(2) z(8) z(9),z(6) z(7) z(3) z(10), z(8) z(9) z(10)
z(4)],

%determino A
[s v]=eigs(B),
A=(sqrt(abs(v))*(s'),


%definisco S ed L
S2=(A*S).*gain,
L2=L*(A)^-1,
L2=L2(:,2:4),

%estraggo da S le componenti della normale
nx=reshape(D(2,:),row,col),
ny=reshape(D(3,:),row,col),
nz=reshape(D(4,:),row,col),


%determino i gradienti e la normale
albedo=sqrt(nx.^2+ny.^2+nz.^2),
p=(-nx./nz),
q=(-ny./nz),
N=albedo.*(p./sqrt(p.^2+q.^2+1)),
N(:,:,2)=albedo.*(q./sqrt(p.^2+q.^2+1)),
N(:,:,3)=albedo.*(1./sqrt(p.^2+q.^2+1)),
p=(albedo.*p),
q=(albedo.*q),
end
```

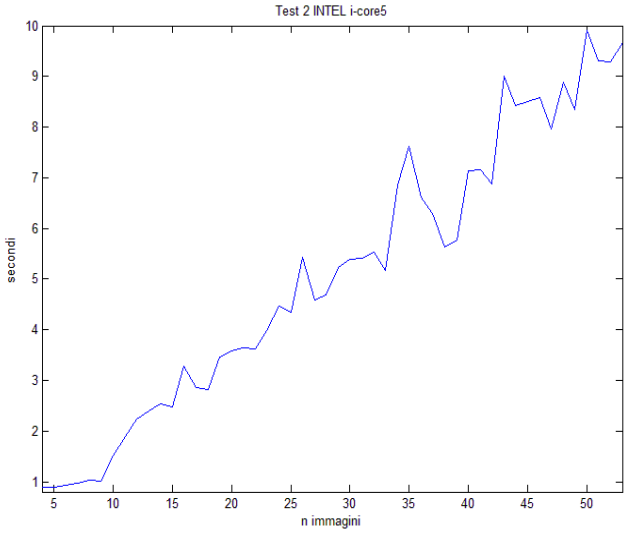
Il grosso carico computazionale è sicuramente rappresentato dalla prima operazione di decomposizione, che comporta processi di fattorizzazione mediante calcolo di autovalori e autovettori e processi di ortogonalizzaione, nel bene o nel male questo ci porta a spendere $\boldsymbol{O}(n^3)$. La scelta però di utilizzare il fattore di troncamento, ci consente in realtà di limitare molto questo fattore, a patto però di ricavare una trasformazione lineare che ci consenta poi di ricavare con precisione $S$. Ciò dipende ovviamente molto dal condizionamento iniziale di $M$ e dalla sua

dimensione. Come per l'algoritmo di ricostruzione con approccio constrained, anche in questo caso se si vogliono ottenere buoni risultati, e quindi minimizzare il rumore, occorre utilizzare un numero di immagini superiore a quattro, a meno che l'oggetto in questione non rispetti le condizioni di Lambertianità, allora in tal caso possono essere sufficienti anche sole quattro immagini. Anche in questo caso per completezza riportiamo alcuni risultati grafici utili per la valutazione dei tempi di calcolo:



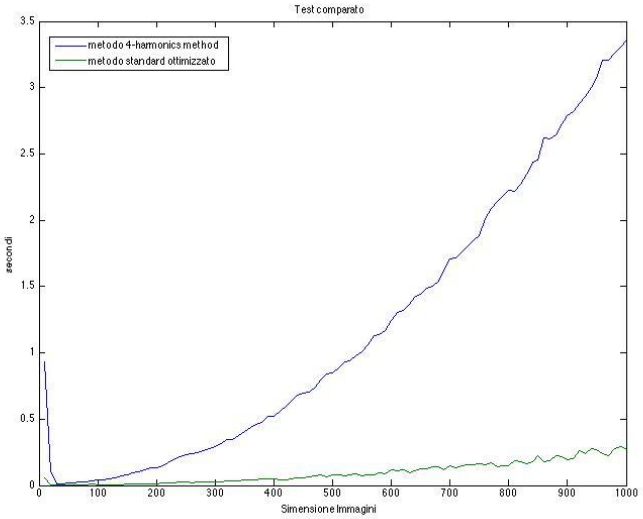



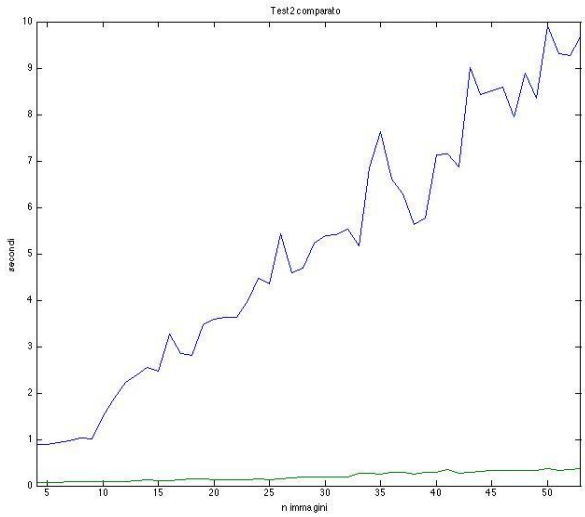




Le prove effettuate sono identiche a quelle precedenti. Ciò che emerge immediatamente dai dati, differentemente con quanto osservato nel metodo standard, è che la dimensione della matrice $M$ incide sicuramente sui tempi di elaborazione, è altresì vero però che cinquanta immagini per data set risulta un numero spropositato. Come abbiamo già detto diverse volte, sei immagini di buona qualità sono già sufficienti per avere risultati discreti. Chiaramente se si confrontano i risultati con quelli precedentemente ottenuti pare che il metodo *four-harmonics-method* sia da scartare a priori. In realtà però è molto più stabile in termini di risultati e soprattutto non richiede la matrice delle direzioni di luce, che come detto precedentemente deve essere costruita con particolare cura e precisione (il che non è sempre un'operazione semplice) se si vogliono ottenere dei gradienti di

media qualità. Per completare il discorso sui metodi di elaborazione dei gradienti, riportiamo un test di confronto tra il metodo standard ottimizzato e il *four-harmonics-method*:



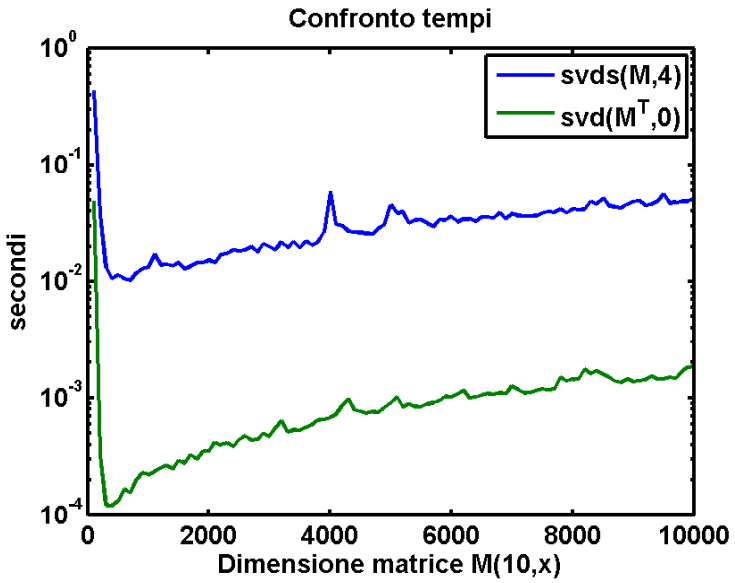

### 2.5.2 ottimizzazione

Nell'implementazione precedente abbiamo utilizzando una decomposizione SVD troncata che è stata implementata in Matlab utilizzando la funzione: $\mathrm{svds}$. In realtà tale funzione si aspetta in ingresso una matrice di dimensione $m \times n$ , in cui sia il numero di righe che di colonne è molto elevato. Però nel nostro caso $M$, la matrice delle immagini, assume una struttura particolare, in quanto il numero di pixel e decisamente maggiore rispetto al numero delle immagini. Pertanto utilizzare la $\mathrm{svds}$ è un operazione non del tutto corretta. Infatti Matlab mette a disposizione la possibilità di calcolare decomposizione SVD in configurazione: *'economy size'* $(\mathrm{svd}(\mathbb{M}, 0))$. Questa è specialmente indicata quando si ha a che fare con matrici di questo tipo, in quanto il numero di valori singolari calcolabile ovviamente coincide con il numero delle immagini. Tutti gli altri valori singolari sono invece identicamente nulli pertanto in modalità *economy size* non vengono calcolate le basi ortonormali corrispondenti a tali valori. Questo risparmio di calcolo, consente di snellire enormemente i tempi di calcolo, però resta ancora un punto da chiarire. Infatti abbiamo definito $M$ come una matrice $n \times row * col$, in cui n rappresenta il numero delle immagini. Se utilizzassimo la decomposizione in *economy size* con la matrice $M$ in questa configurazione riscontreremmo problemi di *out fo memory.* Questo in realtà perché il calcolo, a livello matematico, è concepito su delle matrici rettangolari a colonna, quindi è sufficiente trasporre la matrice $M$. La chiamata risultante sarà pertanto : $\mathrm{svd}(\mathbb{M}', 0)$. A livello implementativo cambia poco, tutto l'algoritmo descritto precedentemente resta valido. L'unica modifica da apportare concerne la definizione stessa delle matrici $\tilde{L}$ e $\tilde{S}$ che a questo punto saranno così definite:

$$\tilde{L} = V_4\sqrt{\Delta_4} \qquad \tilde{S} = \sqrt{\Delta_4}{U^T}_4$$

Ovviamente noi siamo interessati solamente ai primi quattro valori singolari, pertanto anche se in configurazione economy size, non è possibile scegliere un fattore di troncamento, sarà sufficiente ridimensionare $U$ e $V$ in modo tale che $V_4$ sia di dimensione $n \times 4$ e ${U^T}_4$ di dimensione $4 \times row *$

$col$. Per rendervi partecipi dei miglioramenti ottenuti a seguito di tale modifica, riportiamo un test effettuato che confronta i tempi di calcolo utilizzando svds con fattore di troncamento quattro e *l'economy size* decomposition, fissando il numero delle immagini di $M$ pari a 10 e facendo invece aumentare la dimensione delle immagini, quindi di $row * col$.




## 2.6 Banco di prova

In questo paragrafo mostreremo alcuni risultati ottenuti utilizzando i metodi di elaborazione dei gradienti che abbiamo illustrato precedentemente. Pertanto in una prima fase testeremo il metodo di elaborazione standard illustrato nel paragrafo 2.3 e successivamente il metodo di elaborazione *4-harmonics method* trattato in 2.4. Per far ciò faremo uso in un primo momento di data set sintetizzati. Tale operazione è di vitale importanza dal momento che ha come finalità quella di dimostrare che effettivamente gli algoritmi costruiti siano in grado di assolvere al loro compito. Inoltre è un approccio che viene spesso utilizzato in matematica. In altri termini si parte da un problema in cui si conosce la soluzione vera e si cerca di capire, in base ai metodi utilizzati, quanto ci si avvicini alla soluzione reale. In un secondo momento mostreremo qualche ulteriore risultato facendo uso di data set reali cercando inoltre di confrontare i risultati ottenuti.

### 2.6.2 Costruzione del modello sintetizzato per il metodo di elaborazione standard

Prima di procedere con la costruzione del modello sintetizzato per il metodo standard di elaborazione dei gradienti è opportuno introdurre qualche considerazione. Tipicamente quando risolviamo un problema di Photometric Stereo, l'obbiettivo è quello di ricostruire delle superfici a partire da dei data set composti da diverse immagini. Pertanto, come sottolineato più volte durante il testo, la vera incognita è proprio la superficie che abbiamo definito come una funzione in due variabili del tipo $z = f(x, y)$ che ricaviamo a partire dalla conoscenza dei gradienti. Quindi risolviamo sostanzialmente un problema inverso, mentre nel caso della costruzione del modello sintetizzato risolviamo il problema diretto. In altri termini ribaltiamo il problema imponendo una certa $f(x, y)$, che questa volta però conosciamo, da cui possiamo ricavare un data set che poi

daremo in pasto al metodo di elaborazione. Per far ciò possiamo partire dalla relazione (2.14) che riscriviamo per comodità come:

$$\begin{cases} T = R^{-1}(M^T Q)^T \\ L = QR \end{cases}$$

dove questa volta la vera incognita del problema è proprio la matrice delle immagini ordinata in modo lessicografico ovvero: $M$. Quest'ultima può essere ricavata in modo abbastanza semplice invertendo la relazione precedente e quindi:

$$M = (Q^{-1})^T R\, T \qquad (2.33)$$

La (2.32) deve essere verificata per una qualsiasi matrice di direzione $L$ che può essere scelta arbitrariamente, pertanto nota quest'ultima sarà nota la sua trasformazione $QR$. L'unica incognita quindi nella (2.32) è proprio la matrice $T$ che può essere ricavata molto facilmente se si conoscono i gradienti $p\ e\ q$ definiti in (2.3), ovvero come le derivate parziali rispetto a x e y della superficie, che nel caso in esame è nota e di conseguenza lo sono i p e q. Per ricavare $T$ possiamo dunque risolvere un sistema così fatto:

$$\begin{cases} t_1 = -p\ t_3 \\ t_2 = -q\ t_3 \\ albedo = \sqrt{{t_1}^2 + {t_2}^2 + {t_3}^2} \end{cases} \qquad (2.34)$$

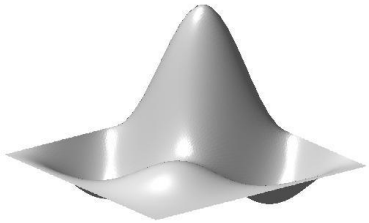
Per rendere (2.34) determinato possiamo assumere l'albedo costante su ogni punto. Tale assunzione è del tutto lecita dal momento che stiamo lavorando su superfici del tutto ideali. Pertanto se risolviamo (2.34) in funzione di $t_3$ otteniamo :

$$t_3 = \frac{cost}{\sqrt{p^2 + q^2 + 1}} \qquad (2.35)$$

una volta noto $t_3$ è immediato ricavare per sostituzione diretta in (2.34) sia $t_1$ e $t_2$ e quindi possiamo costruire $T$ semplicemente incolonnando i vettori riga precedentemente ottenuti. A questo punto il problema è del tutto determinato in quanto conosciamo tutti gli elementi che compongono (2.33) da cui possiamo finalmente costruire i nostri data set sintetici.

La scelta della $z = f(x, y)$ non è assolutamente banale, soprattutto in quei casi in cui le superfici sono geometricamente complesse. Infatti quest'ultime possono avere dei campi di esistenza e quindi possono essere centrate all'interno di un dominio. Questa purtroppo è un informazione che perdiamo nel momento in cui determiniamo $M$. Pertanto la scelta migliore è quella di costruire delle $f(x, y)$ prive di campi di esistenza o comunque definite su un intervallo di ampiezza $0 - n$ e, dove n rappresenta quindi il numero di pixel di ogni singola immagine. Per far ciò si può pensare di utilizzare delle funzioni trigonometriche e combinarle in modo tale da costruire delle superfici, possibilmente non banali e prive di simmetria in modo tale da metterci nella condizione più generale possibile. La $f(x, y)$ che analizzeremo è la seguente:

$$z = xy\,[\sin(\pi x)\sin(\pi y)] \qquad (2.36)$$

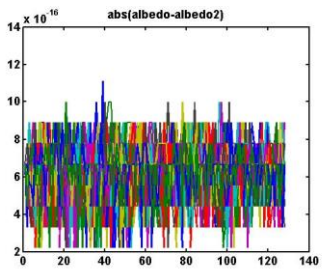


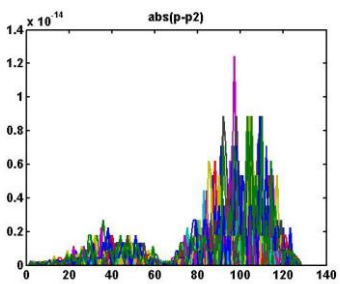
Come prima cosa dobbiamo determinare la matrice $T$ che costruiamo a partire dalla conoscenza dei gradienti. Per far ciò è sufficiente dunque derivare rispetto a x e rispetto a y la (2.36) ottenendo:

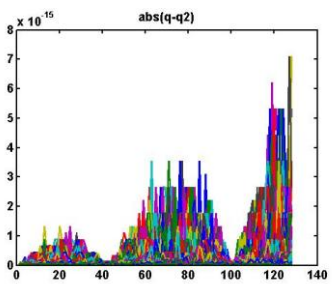
$$\begin{cases} p = \dfrac{\partial z}{\partial x} = ysin(\pi y)[\sin(\pi x) + \pi x cos(\pi x)] \\ q = \dfrac{\partial z}{\partial y} = xsin(\pi x)[\sin(\pi y) + \pi y cos(\pi y)] \end{cases} \qquad (2.37)$$

risolvendo prima (2.34) e poi (2.35) finalmente otteniamo la matrice $M$. La sua dimensione ovviamente dipenderà dalla scelta di $L$. Pertanto ipotizzando di illuminare la nostra superficie in quattro direzioni differenti avremo come risultato la sintesi di quattro immagini differenti che possiamo estrarre da $M$ e visualizzare. Se per esempio costruissimo $L$ riferendoci ai quattro punti cardinali si ottengono le seguenti immagini:

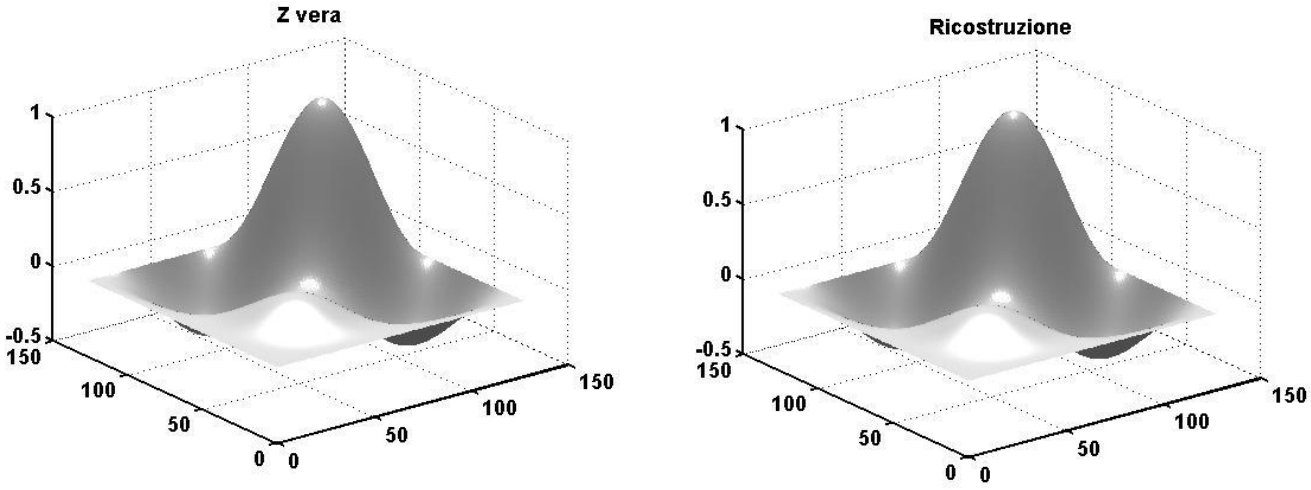
A questo punto possiamo passare la matrice $M$ , come parametro di ingresso, alla funzione che abbiamo descritto in 2.3 che ci restituirà in uscita i gradienti e l'albedo, questa volta però risolvendo il problema fotometrico e quindi affrontando il problema in modo inverso. Una volta determinati i gradienti questi ultimi possono essere confrontati con quelli veri definiti da (2.37). Allo stesso modo possiamo confrontare anche gli albedo, che ricordiamo sono delle costanti. Sarà sufficiente calcolare una sottrazione punto per punto per verificare che effettivamente i parametri sintetizzati coincidano con quelli ricavati a partire da $M$. Questo è ciò che si ottiene:



Gli albedo come ci aspettavamo coincidono perfettamente, infatti variano su una scala di valori dell'ordine $10^{-16}$ che coincide con l'errore di macchina (eps). Allo stesso modo anche i gradienti sono identici quest'ultimi variano su una scala che è compresa tra $10^{-14}, 10^{-15}$ . Questi risultati sono sicuramente significativi in quanto ci consentono di affermare che il metodo di elaborazione standard effettivamente risolve il problema fotometrico e se si conoscono le direzioni della luce reali, i risultati che si ottengono sono praticamente identici, a meno dei limiti di memorizzazione dei moderni calcolatori. Il test finale consiste nella ricostruzione 3-D a partire dalla conoscenza dei gradienti, per far ciò si risolve un problema di integrazione, che avremo modo di approfondire nel capitolo seguente. Pertanto se pur non si tratti di un'operazione necessaria, in quanto aver dimostrato che i gradienti coincidano è già sufficiente ai fini di questo paragrafo. Preferiamo comunque mostrarvi per completezza e per meglio testimoniare la qualità dei risultati, qualche risultato grafico:

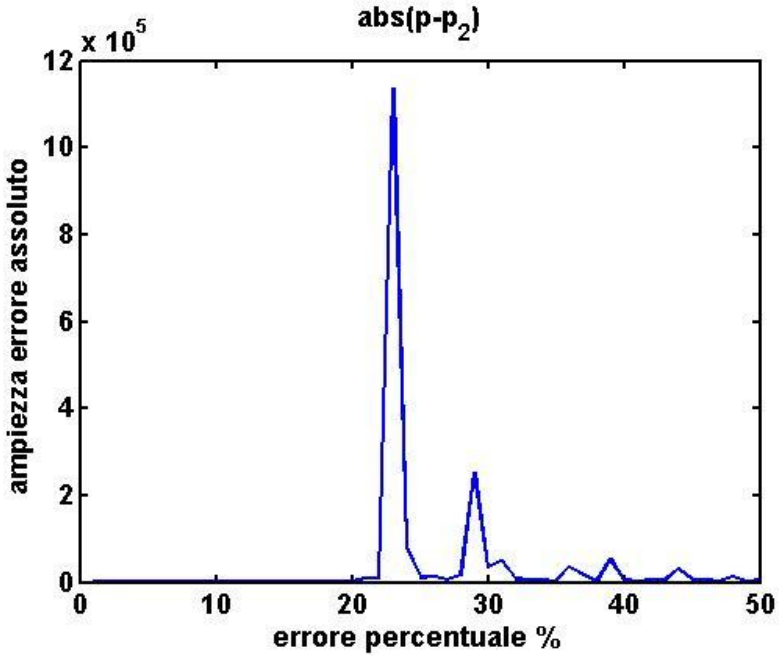



E' immediato constatare come almeno graficamente i risultati che si ottengono sono identici. In realtà se andiamo ad effettuare la differenza delle due superfici si attesta uno scarto medio del 0.5 % che in realtà però è dovuto non tanto all'effetto della distorsione dei gradienti, che è dell'ordine di $10^{-14}$, ma al processo di integrazione, che è comunque un risultato più che lecito.

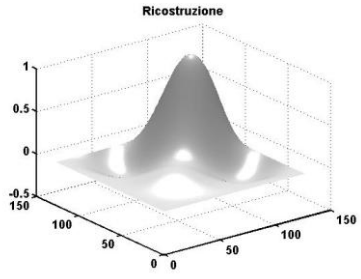
A questo punto una volta dimostrata la validità del metodo di elaborazione standard dei gradienti possiamo passare allo step successivo che consiste nello studio della robustezza del metodo stesso. Per esempio potremmo domandarci quali sono gli effetti che entrano in gioco in presenza di eventuale rumore. Quest'ultimo può manifestarsi in diverse forme e coinvolgere differenti componenti. Infatti più volte abbiamo sottolineato l'importanza della conoscenza precisa della matrice $L$ per ottenere ricostruzioni affidabili. Pertanto in questa sede cercheremo di studiare quali sono gli effetti che scaturiscono a seguito di piccole variazioni delle direzioni luce. Un'altra componente, che nei casi reali è afflitta da disturbo, è sicuramente la matrice delle immagini $M$. Nel test precedente ovviamente $M$ era priva di rumore dal momento che è stata determinata direttamente a partire dal modello sintetico. Ma nella realtà dei fatti M è una matrice che viene fuori da delle immagini di tipo digitale, che sono a tutti gli effetti dei dati campionati e soggetti a un processo di campionamento e quantizzazione in cui parte dell'informazione viene persa. Non solo

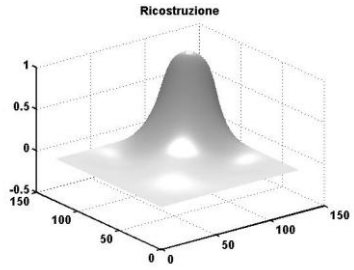
ma tali immagini vengono inoltre compresse per limitare lo spazio occupato in memoria con conseguente perdita di ulteriore informazione che di fatto si traduce in rumore. Cercheremo a questo punto di effettuare dei test mirati, con la finalità di tracciare dei range di applicabilità del metodo.

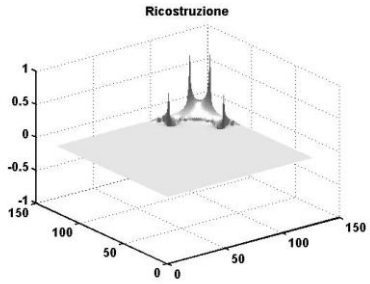
Partendo con ordine, come prima cosa abbiamo perturbato la matrice delle direzioni di luce, aggiungendo dell'errore che abbiamo aumentato gradualmente dall'uno per cento sino ad arrivare al cinquanta. Dopo di che la matrice perturbata è stata utilizzata per la determinazione dei gradienti fotometrici che verranno confrontati con quelli reali stimando l'errore assoluto tra le norme infinito delle due grandezze. Di seguito riportiamo il risultato del test:



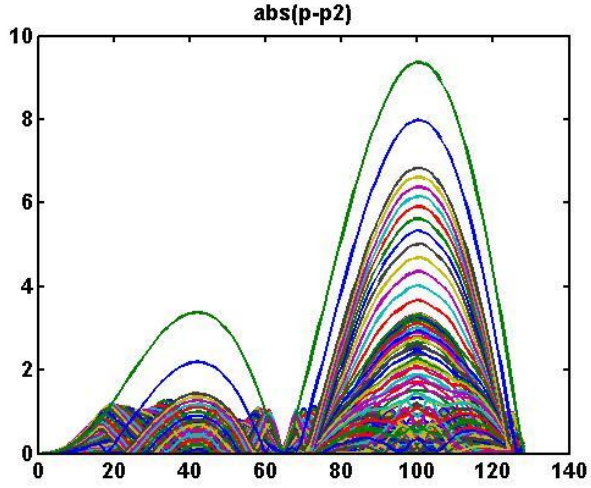

Sino al 20% di errore sembrerebbe che lo scostamento misurato in termini assoluti si mantenga pressoché stazionario. In realtà però ci troviamo su una scala di ordine $10^5$, quindi lo scostamento incomincia ad essere significativo già per valori superiori al 15%. Inoltre dal 20% in poi si registra un comportamento oscillatorio in cui si raggiungono picchi piuttosto elevati che tendono nuovamente ad annullarsi al crescere dell'errore. Tale andamento è sostanzialmente giustificato dal fatto che abbiamo aggiunto un errore costante su tutte le componenti della matrice $L$. Nella pratica questo non è del tutto vero perché l'errore per esempio potrebbe interessare una sola delle componenti di $L$. Il vero banco di prova spetta comunque al risultato che si ottiene dalla ricostruzione della superficie, di seguito riportiamo tre ricostruzioni effettuate con i gradienti perturbati al quindici, venti e trenta per cento.

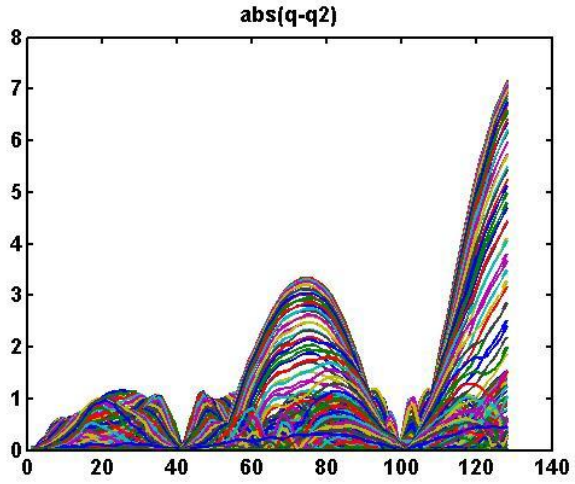


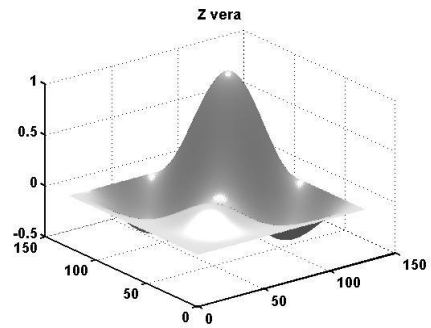



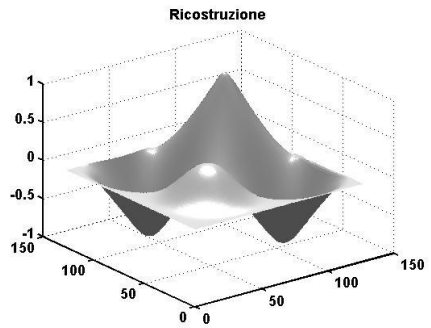
Come si può osservare se ci si mantiene sotto uno scostamento del 15% si riescono ad ottenere ancora delle ricostruzioni piuttosto fedeli che non si allontanano di molto rispetto alla soluzione vera. Dal venti in poi le ricostruzioni incominciano ad essere meno fedeli sino al trenta in cui in realtà non si riesce più a costruire alcuna superficie. Nella realtà dei fatti raggiungere il 30% di errore non è poi così difficile ed è comunque ancora possibile in quelle circostanze ricostruire delle superfici attendibili. Questo perché nel modello sintetico abbiamo considerando delle sorgenti all'infinito, che è del tutto una condizione di idealità, mentre nelle acquisizioni reali questa specifica non può essere replicata. Sembrerebbe , dai numerosi risultati osservati, che tale fattore di non idealità interferisca costruttivamente nella riduzione dell'errore stimato sulle direzioni luce, il che ovviamente gioca a favore dell'applicabilità del metodo. Una volta quindi studiati gli effetti dovuti alla modifica della matrice delle direzioni luce, siamo passati a esaminare gli effetti dovuti alla perturbazione della matrice $M$. In accordo con quanto detto precedentemente questa può subire forti cambiamenti a seguito dei processi di compressione immagine che provocano quindi perdita di informazione. In questo caso piuttosto che perturbare direttamente $M$ aggiungendo del rumore abbiamo preferito metterci nella condizione reale. Ovvero abbiamo convertito le immagini del modello sintetizzato in formato jpg, e le abbiamo utilizzate quindi per rideterminare i gradienti. Anche in questo caso abbiamo imbastito lo stesso banco di prova rivalutando lo scostamento in termini di valore assoluto tra i gradienti ottenuti e quelli reali (2.37).




I gradienti che si determinano sono sicuramente differenti come ci aspettavamo. Se inoltre si calcolano le norme infinito, quest'ultime superano facilmente il valore di duecento. In realtà dai grafici pare che compaia qualche fattore di scala giustificato da una forte componente di addensamento per valori di ordinata compresi tra zero e uno. Nuovamente riproponiamo, per un confronto diretto, la superficie vera con quella ottenuta a partire dai gradienti risultanti dalle quattro immagini compresse.



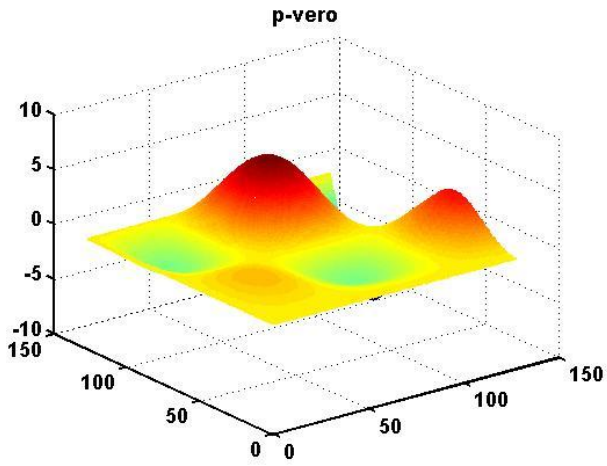
In realtà anche in questo caso il risultato pare essere piuttosto severo rispetto ai numerosi test effettuati su data set reali, sebbene lo shape ricostruito sia più che coerente con quello di partenza. Inoltre la dimensione delle immagini gioca sicuramente un ruolo importante. In questo caso abbiamo lavorato su dimensioni di $128 \times 128$, quindi abbastanza piccole, ciò ci porta a pensare che l'effetto della perdita di informazione sicuramente decresce con l'aumentare della dimensione delle immagini. In conclusione, dall'analisi dei test effettuati sul metodo di elaborazione standard, siamo in grado di poter affermare che quest'ultimo presenta una certa stabilità e solidità se i parametri di ingresso risultano poco perturbati. Questo rappresenta poi di fatto il vero punto di forza rispetto ad altri metodi, motivo per il quale è stato e continua tuttora ad essere uno dei più applicati in ambito Photometric Stereo.

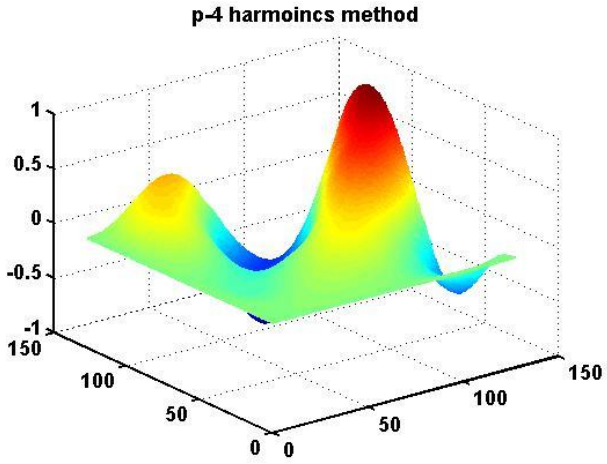
### 2.6.3 Test sul metodo di elaborazione 4-harmonics-method (uncosntraint)

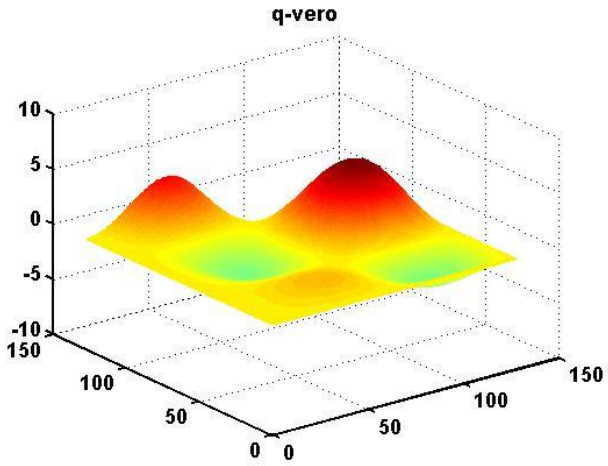
In questo sotto paragrafo riporteremo alcuni risultati ottenuti con il metodo di elaborazione dei gradienti: *4 harmonics-method.* Contrariamente con quanto visto in precedenza, non costruiremo un modello sintetizzato. Questo perché non solo tale processo richiederebbe parecchio tempo ma assume una complessità abbastanza elevata, che presuppone la conoscenza di alcune componenti a noi purtroppo mancanti. Sarà sicuramente nostra premura però cercare meglio di capire in studi futuri come poter impostare il problema. In realtà uno dei test che però abbiamo effettuato con il modello sintetizzato, è stato quello di ricavare la trasformazione di *Lorentz* a partire dalla conoscenza della matrice delle armoniche sferiche $S$ che può essere totalmente ricavata a partire dalla conoscenza della $f(x, y)$. Per far ciò abbiamo semplicemente invertito la relazione (2.20) che quindi abbiamo riscritto come: $A^T = (S\,\tilde{S})^{-1} S^T$, dove ricordiamo $\tilde{S}$ viene fuori dalla decomposizione a valori singolari della matrice $M$. Successivamente non abbiamo fatto altre che confrontare la trasformazione vera, con quella ottenuta utilizzando l'algoritmo descritto in 2.5. I risultati non ci hanno sorpreso più di tanto in quanto se dividiamo termine per termine le rispettive componenti non otteniamo un fattore di scala costante. In realtà abbiamo avuto modo di verificare che ciò dipende dal fatto di costruire la matrice $Q$ a partire dalla matrice $\tilde{S}$, e non $S$ che ovviamente nell'applicazione diretta del metodo è la vera incognita del problema. Queste due matrici sono sicuramente diverse, e lo sono altrettanto il loro spazi di nullità, che abbiamo utilizzato per ricavare la matrice $B$. Ciò ci suggerisce che in tal senso forse si potrebbe fare qualcosa di meglio, come per esempio piuttosto che applicare una decomposizione per valori singolari e cercare un'opportuna trasformazione lineare. Per far ciò si potrebbe pensare di applicare direttamente una decomposizione per armoniche sferiche semplicemente sviluppando in serie la matrice $M$.

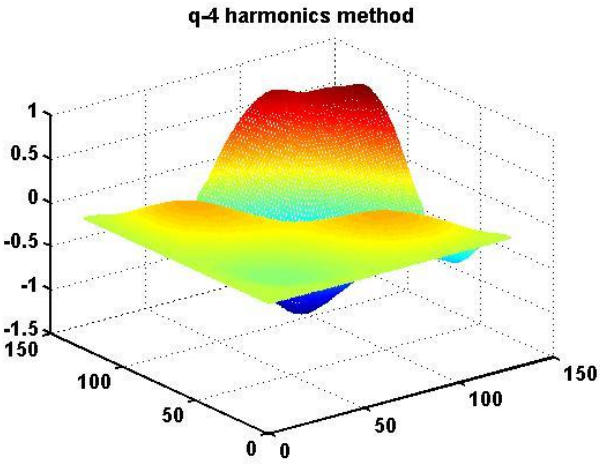
A meno comunque di questa forte ambiguità, e l'incapacità di utilizzare un modello sintetizzato opportuno, abbiamo comunque ottenuto degli ottimi risultati per il quale vale la pena analizzare. Il primo test che abbiamo effettuato è stato quello di utilizzare la matrice $M$, ricavata dal modello sintetizzato precedente, che abbiamo passato come parametro di ingresso all'algoritmo descritto in 2.5. Si osservi la vera differenza rispetto a quanto visto in precedenza, tale metodo non ha bisogno di conoscere a priori le varie direzioni della luce che in realtà dalla costruzione del modello

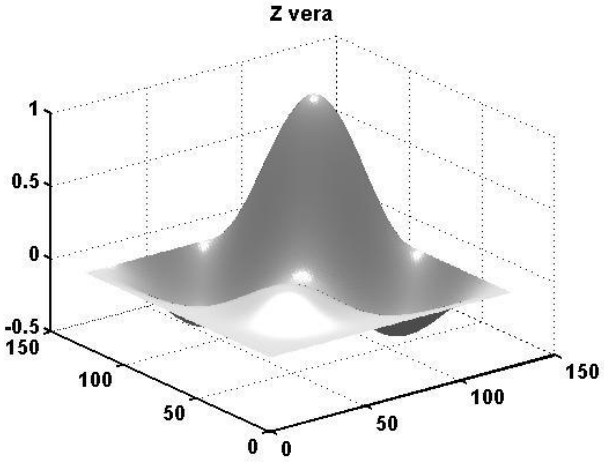
sintetizzato noi conosciamo perfettamente. Differentemente con quanto fatto in precedenza in questo caso non mostreremo la differenza tra il gradiente vero è quello elaborato, se non fosse che sono completamente differenti in accordo con quanto detto poco sopra. Piuttosto preferiamo darvi un riferimento grafico, mostrandovi direttamente le elaborazioni ottenute e la ricostruzione del modello 3-D risultante, che come detto in precedenza, è poi di fatto il vero indicatore sulla qualità dei gradienti elaborati:

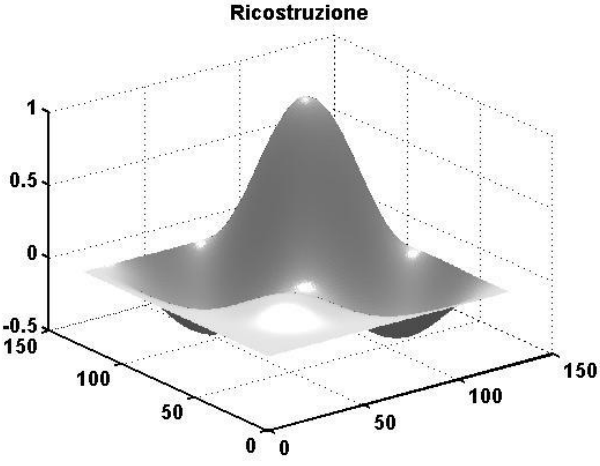


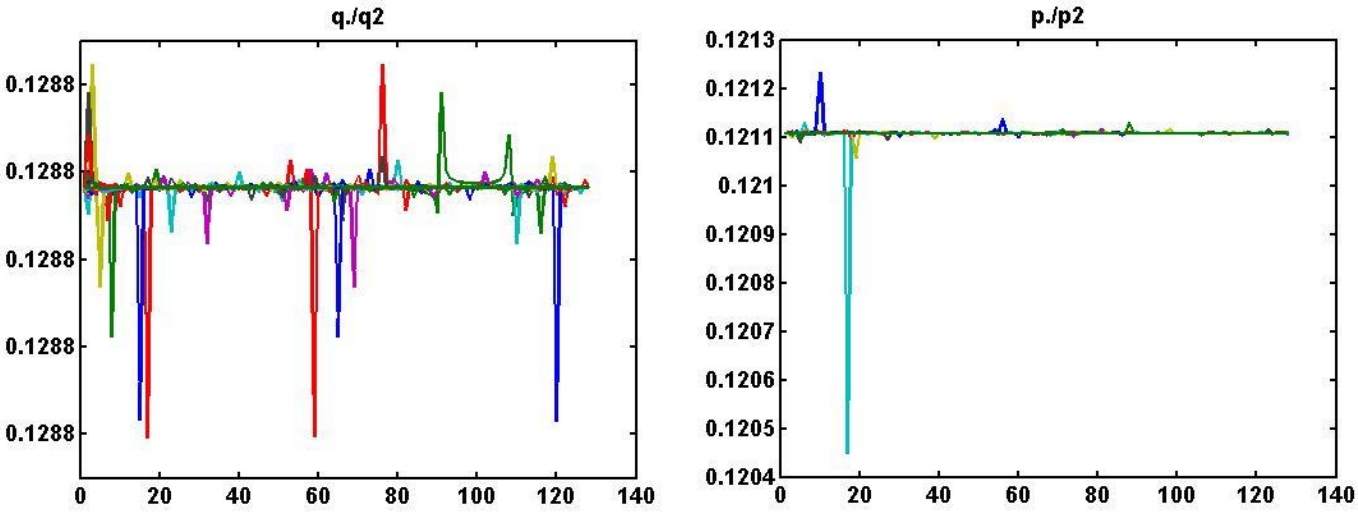



Le due figure sono anche in questo caso identiche. Se inoltre si sottrae la superficie ottenuta con quella vera si ottiene uno scostamento che è inferiore al dieci per cento, a cui va sottratto l'errore dato dal processo di integrazione. Contrariamente i profili grafici dei gradienti, sono completamente differenti, nonostante ciò le ricostruzioni sono comunque fedeli rispetto al modello di partenza e questo perché il processo di integrazione ha come effetto quello regolarizzare e quindi di smorzare gli errori. Un altro esperimento interessante è stato poi quello di utilizzare la matrice delle direzioni luce, derivante dal *4-harmonics,* per elaborare i gradienti con il metodo standard. In questo caso per il confronto, dal momento che comparivano fattori di scala, abbiamo preferito esprimere in termini di rapporto, componente per componente e plottare il risultato. Questo è ciò che si ottiene:

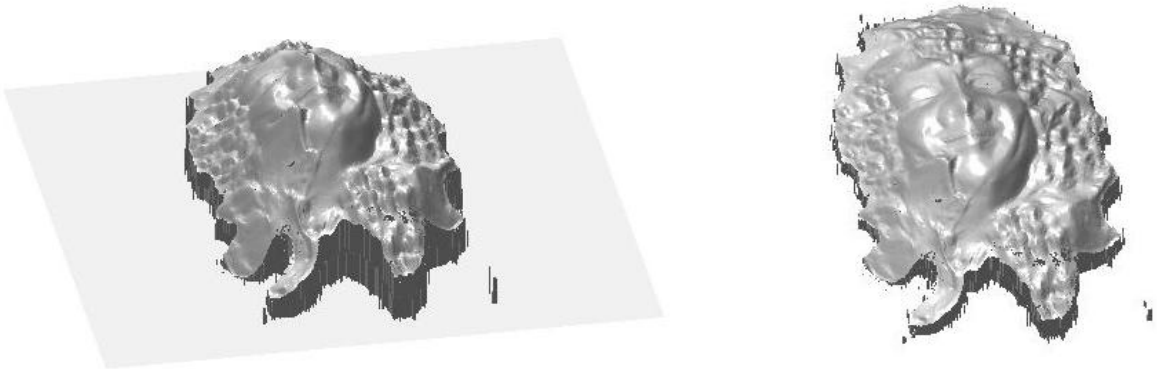



Come si può constatare, a meno di qualche scostamento che rimane comunque inferiore al dieci per cento, il rapporto è costante. Il che vuol dire che a meno di in fattore di scala i gradienti che si ottengono sono identici in ambo i casi. Rimane però a questo punto una contraddizione molto forte, in quanto la matrice $M$ che abbiamo utilizzato, è stata costruita utilizzando quattro immagini in cui la superficie viene illuminata nelle quattro direzioni cardinali, pertanto ci saremo aspettati di ottenere una matrice $L$ coerente con le direzioni fissate, ma purtroppo così non è. Nonostante ciò questo fatto sembra non pregiudicare i risultati, aspetto che sicuramente merita più attenzione e ulteriori approfondimenti, che esulano però dagli obbiettivi di questo lavoro.

Dopo questi primi test iniziali che ci hanno consentito fondamentalmente di dimostrare la funzionalità del metodo, passiamo ad applicarlo a un caso reale. In questo caso non faremo dei test di robustezza per il semplice fatto che faremo uso dati reali, che oltre a non rispettare condizioni di idealità, sono fortemente affetti da errore a seguito dei vari processi di acquisizione e compressione immagine. Per far ciò abbiamo costruito un data set utilizzando una semplice webcam e un flash di un telefono cellulare con i quali abbiamo ricavato otto diversi fotogrammi. In accordo al metodo, non ci siamo ovviamente preoccupati di posizionare la sorgente luminosa in punti prefissati, che al contrario è stata posizionata su dei punti puramente casuali. Di seguito riportiamo le otto immagine acquisite:

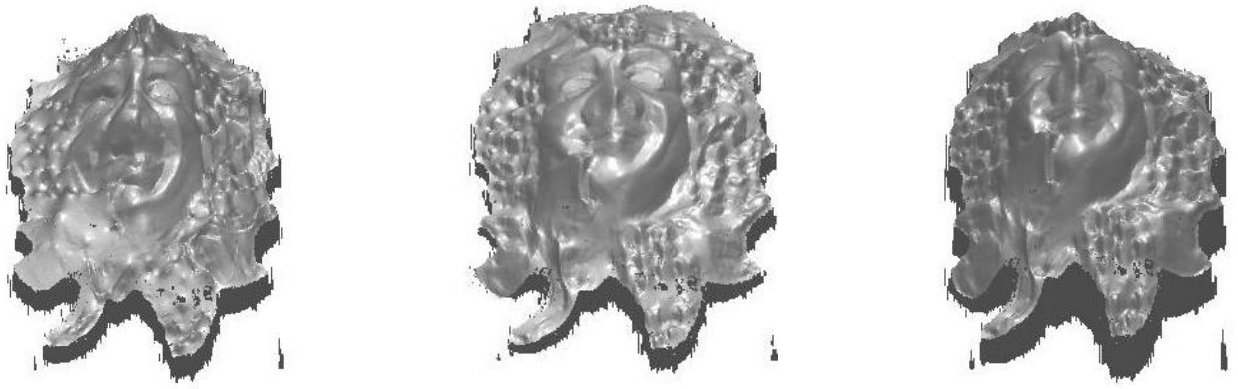
Una volta acquisite le immagini possiamo costruire la matrice $M$ e passarla come parametro di ingresso all'algoritmo, ottenendo quindi in ordine: l'albedo i gradienti e la normale punto per punto:

Integrando i gradienti quindi finalmente si ottiene la ricostruzione della superficie riportata nelle due figure sottostanti:



Le ricostruzioni sembrano essere piuttosto fedeli, a meno di qualche spot che in realtà è dovuto dalla condizione al contorno necessaria per la fase di integrazione, il che ovviamente ci fa pensare che l'elaborazione dei gradienti è stata fatta in maniera piuttosto accurata e precisa. Anche in questo caso se utilizziamo la matrice $L$ risultante per poi determinare i gradienti col metodo di elaborazione standard, si ottengono i medesimi risultati, in rafforzamento a quanto mostrato in precedenza. Successivamente ci siamo domandati quale sarebbe stato il risultato riducendo il

numero delle immagini e quindi riducendo la dimensione della matrice $M$. Quindi in ordine siamo partiti ricostruendo la superficie utilizzando solo quattro immagini, e poi gradualmente aumentando il numero a cinque e infine a sette.

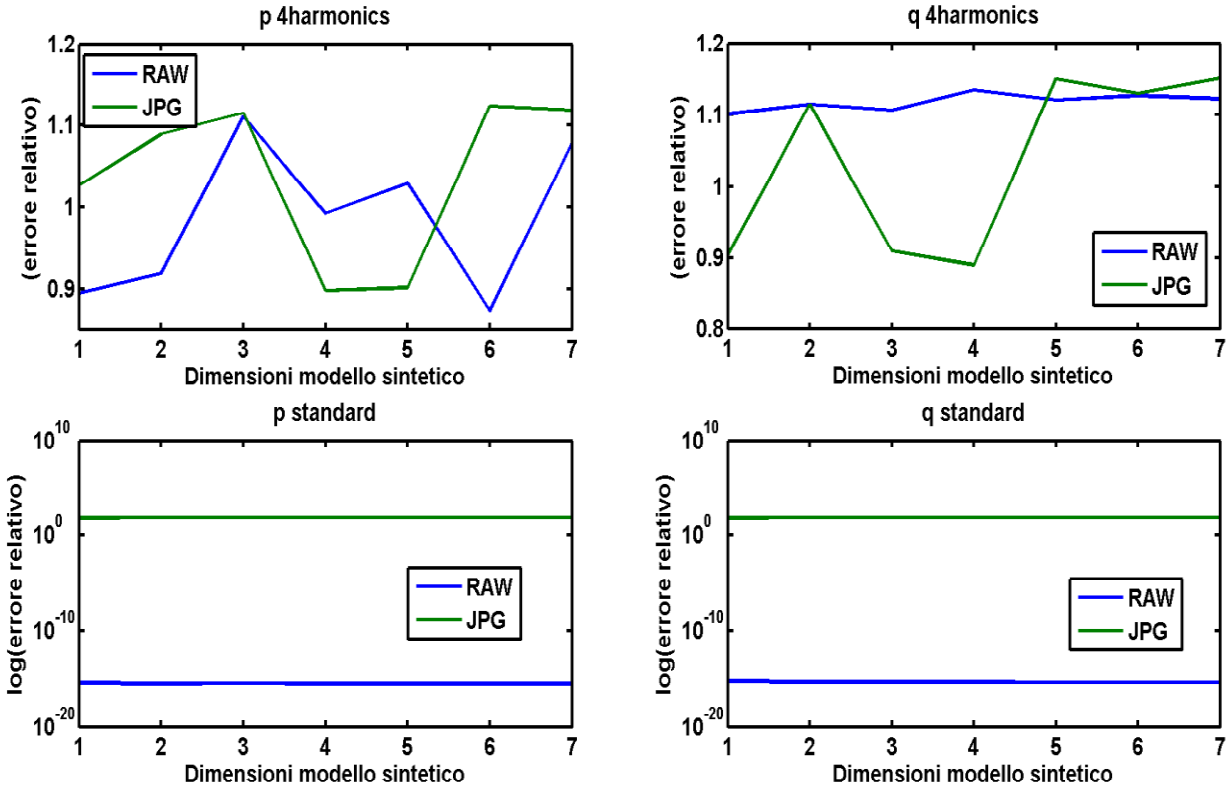


Con sole quattro immagini, come è constatabile dalla figura, la ricostruzione avviene in modo piuttosto distorta. Il che implica che il numero minimo di immagini non è sempre sufficiente per garantire le informazioni necessarie in merito all'elaborazione dei gradienti. Contrariamente con cinque immagini la ricostruzione è molto più affidabile per arrivare sino a sette in cui invece la ricostruzione è praticamente la medesima a quella ottenuta nel caso di utilizzo di tutte le otto immagini. Tale risultato ovviamente ci consente di poter affermare che è comunque possibile ottenere delle ricostruzioni fedeli senza avere a che fare con problemi troppo sovradeterminati che effettivamente può incidere sui tempi di calcolo, soprattutto se quest'ultime assumono dimensioni importanti. Indicativamente un buon compromesso tra qualità dei risultati e tempi di calcolo consiste nell'utilizzare data set composti da sei immagini. In tutti questi casi abbiamo ottenuto risultati sempre più che soddisfacenti. In realtà un piccolo difetto che abbiamo constatato emerge nella determinazione dei segni dei gradienti. Dalle nostre analisi questo fatto dipende proprio dalla decomposizione ai valori singolari e nella determinazione della matrice $\tilde{S}$ che in prove ripetute, su medesimi data set, ha mostrato dei cambi di segno inaspettati. Abbiamo anche avuto modo di constatare però che questo fenomeno si attenua drasticamente con l'aumentare della dimensione della matrice delle immagini.

In conclusione, dai numerosi test effettuati che purtroppo non possiamo riportare, siamo in grado di poter affermare che il metodo è piuttosto valido. Non solo ma risulta essere molto più preciso rispetto al metodo di elaborazione standard, soprattutto in tutti quei casi in cui non si è in grado di poter stimare con precisione le direzioni della luce. E'altresì vero che il metodo di elaborazione standard è imbattibile in termini di tempo di calcolo, ma in realtà questo non ci spaventa più di tanto soprattutto se si pensa di utilizzare metodi di integrazione altamente performanti che consentano di colmare il gap. Queste pertanto, secondo il nostro punto di vista, sono delle buone motivazioni per continuare a studiare e cercare di ottimizzare tale metodo.

### 2.6.4 Test finale di comparazione

Fin'ora ci siamo limitati ad effettuare dei test mirati su ciascun metodo di elaborazione discusso, cercando il più possibile di estrapolare delle conclusioni qualitative. Per completezza proponiamo un test, più di carattere quantitativo, che mette a confronto i due metodi in relazione all'utilizzo di dati sintetici, quindi in formato RAW, e dati compressi in formato JPG. Il banco di prova anche in questo caso è stato costruito facendo uso del modello sintetico. Pertanto non abbiamo fatto altro che elaborare i gradienti utilizzando prima i dati veri, RAW, e successivamente quelli compressi in JPG, esprimendoli in termini di errore relativo stimato in norma di Frobenius a partire dalla conoscenza dei gradienti veri. Il tutto è stato fatto procedendo iterativamente per sette volte e quindi facendo aumentare la grandezza delle immagini gradualmente(16-32-128-256-512-1024).




Da una prima analisi sembrerebbe che la dimensione delle immagini non incida sui risultati. Il che significa che indipendentemente dalla risoluzione, l'errore che si ottiene ha sempre lo stesso ordine di grandezza. Se confrontiamo i gradienti ottenuti con il 4-harmonics emerge un dato molto interessante. Infatti il fatto di utilizzare dati sintetici o compressi, sembrerebbe non comportare alcuna differenza in termini di errore, che in ambo i casi è piuttosto elevato e ciò dipende dal fatto di non essere in grado di stimare con precisione $A$. Per quanto riguarda il metodo standard l'utilizzo di dati reali o compressi ha un effetto ben più evidente sull'errore. Se si utilizzano i dati sintetici l'errore coincide con la precisione macchina, il che vuol dire che i gradienti coincidono come mostrato in 2.6.3. Ma se si utilizzano immagini compresse l'errore cresce drasticamente di ben diciassette ordini di grandezza. In conclusione se con il 4-harmincs non sussiste alcuna differenza nell'impiego di dati compressi o reali, nel caso del metodo di elaborazione standard i dati devono essere necessariamente in formato RAW se si vogliono ottenere delle ricostruzioni fedeli.

# 3 Ricostruzione della superficie

## 3.1 Introduzione

Il secondo step, una volta determinato il campo dei gradienti $p$ e $q$, consiste nella ricostruzione vera e propria della superficie $z$. Per far ciò si risolve un problema di integrazione, infatti con i gradienti, non abbiamo fatto altro che associare ad punto un valore di quota. In altri termini abbiamo recuperato una mappa delle profondità. I gradienti però ovviamente sono nel nostro caso delle funzioni discrete, pertanto integrando non facciamo altro che unire ogni singolo punto a una corrispondente quota che ci consente quindi alla fine di poter visualizzare la superficie in modo corretto. Agli albori della Photometric Stereo il processo veniva eseguito applicando un algoritmo che faceva forte uso dell' integrazione Riemaniana, ma quest'ultimo risultava molto sensibile al rumore. Non sarà scopo di questo testo presentarli tutti e analizzarli, rimandiamo pertanto a testi più specifici tutti coloro interessati ad avere una panoramica completa ed esaustiva. Nel voler però rimanere fedeli a ciò che è stato già sviluppato in questa e in altre tesi precedenti [5], ci contreremo sullo studio e nell'ottimizzazione di un metodo di integrazione utilizzato dal Prof. Vanzi e l'ing. Pintus che fa uso dell'equazione differenziale di Poisson. L'utilizzo di tale modello differenziale, come vedremo in seguito, assicura una serie di ottimi vantaggi, sia dal punto di vista implementetativo che prestazionale. Prima però di addentrarci nell'analisi vera e propria del modello è nostro interesse dare una breve panoramica di alcuni elementi matematici che ci consentiranno di poter sviluppare il modello e giustificare i risultati ottenuti.

## 3.2 Metodo delle differenze finite

Il metodo delle differenze finite consiste nell'approssimare il valore della derivata di una funzione $u$ in un punto $x_0$, (per il quale bisognerebbe conoscere gli infiniti valori che $u$ può assumere in un intorno di $x_0$) ,con un espressione che tenga conto di un numero finito di valori. In altri termini si passa dall'operazione di limite a quella di rapporto incrementale. Questo é un grosso vantaggio in termini di calcolo, perché consente di trattare un problema in cui compaiono equazioni con derivate parziali come se si trattasse di un sistema lineare del tipo $Ax = \boldsymbol{b}$, in cui $A$ può essere espressa mediante una matrice sparsa, i cui valori dipenderanno dal grado di approssimazione delle derivate. In realtà oltre alla motivazione puramente computazionale, esiste anche una motivazioni fisica. Infatti quando si ha a che fare con processi di acquisizione dati (audio o immagini) difficilmente si ha a che fare con $u$ continue per via dei processi di quantizzazione e campionamento, inoltre disponiamo di architetture finite che ci consentono di memorizzare un numero finito di "valori". Per tali motivazioni è evidente che il concetto di limite viene meno per violazione del principio di continuità. In altri termini il metodo ci consente di poter calcolare le derivate anche se la nostra $u$ non è puramente continua ma bensì campionata. Ciò comporterà un errore ε che può essere però facilmente controllato nella scelta opportuna dell'ordine di

approssimazione ed è in funzione del grado di precisione dei dati che si vogliono elaborare. Per determinare le approssimazioni alle differenze finite possiamo sviluppare in serie di Taylor $u(x)$ nel punto $x_0$, riscrivendo le nostre derivate sotto forma di polinomi. Preso un $h > 0$ e appartenente all'insieme dei numeri reali, possiamo scrivere i seguenti sviluppi fermandoci al quarto ordine:

$$u(x_0 + h) = u(x_0) + hu'(x_0) + \frac{1}{2}\, h^2 u''(x_0) + \frac{1}{6} h^3 u'''(x_0) + O(h^4) \quad (3.1)$$

$$u(x_0 - h) = u(x_0) - hu'(x_0) + \frac{1}{2}\, h^2 u''(x_0) - \frac{1}{6} h^3 u'''(x_0) + O(h^4) \quad (3.2)$$

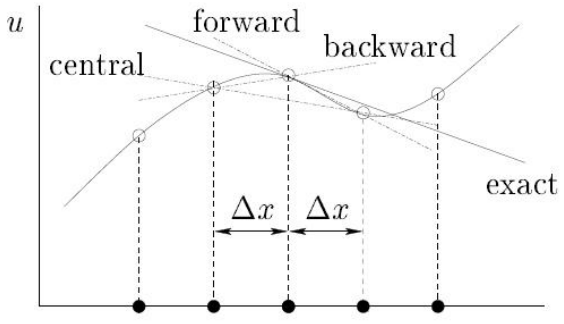
A questo punto sottraendo dalla (3.1) la (3.2) e dividendo per *2h* otteniamo:

$$D_c\, u(x_0) = \frac{u(x_0 + h) - u(x_0 - h)}{2h} = u'(x_0) + O(h^2) \quad (3.3)$$

La (3.3) rappresenta un'approssimazione della derivata prima di secondo ordine. Per le approssimazioni di ordini superiori si può fare riferimento alla combinazione degli operatori differenziali discreti, ma nella maggior parte delle applicazioni l'approssimazione (3.3) è già sufficiente per ottenere buoni risultati, ovviamente maggiore sarà l'ordine $p$ dell'approssimazione, minore sarà l'errore ε che commettiamo in quanto $h$ decresce come un infinitesimo di ordire pari a $h^p$ e tenderà a zero molto più rapidamente.

## 3.3 Interpolazione

Fin ora abbiamo trattato il problema delle differenze finite applicando concetti dell'analisi matematica classica. In realtà per avere degli schemi più generali si è soliti derivare un opportuno interpolatore della funzione $u(x)$. A seconda del tipo di interpolatore scelto, gli schemi che si ottengono sono differenti. Nel nostro caso faremo riferimenti alle formule in avanti (forward), all'indietro(bacward) e centrali (central).




Facendo riferimento alla figura e agli schemi delle differenze finite che derivano dall'operatore Lagrangiano, possiamo scrivere le seguenti relazioni:

$$T_1 = \left(\frac{\partial u}{\partial x}\right)_i \approx \frac{u_{i+1} - u_i}{\Delta x} + O(\Delta x^2) \quad (3.4)$$

$$T_2\left(\frac{\partial u}{\partial x}\right)_i \approx \frac{u_i - u_{i-1}}{\Delta \mathrm{x}} + O(\Delta x^2) \qquad (3.5)$$

Dove $T_1$ e $T_2$ rappresentano rispettivamente le rette tangenti alla curva in configurazione *forward* e *backward.* Per ricavare la *central* è sufficiente sommare la (3.4) e la (3.5) e moltiplicare per un mezzo ricavando in questo modo:

$$T_1 + T_2 = \left(\frac{\partial u}{\partial x}\right)_i \approx \frac{u_{i+1} - u_{i-1}}{2\Delta x} + O(\Delta x^2) \quad (3.6)$$

La (3.4), la (3.5) e la (3.6) rappresentando le approssimazioni alle differenze finite del primo ordine ottenute mediante interpolazione grafica, è immediato constatare inoltre la somiglianza della (3.4) con la (3.6). Il altri termini siamo passati da un'espressione continua a una discretizzata, il che ci porta a scrivere:

$$u(x) \approx u_i \quad u(x+h) \approx u_{i+1} \quad u(x-h) \approx u_{i-1} \quad (3.7)$$

In questo caso può essere interessante ricavare almeno un' approssimazione di ordine superiore limitandoci al solo caso della *central*. L'operazione è piuttosto banale infatti è sufficiente derivare nuovamente la (3.6) ottenendo in tal modo la relazione seguente:

$$\left[\frac{\partial u}{\partial x}\left(\frac{\partial u}{\partial x}\right)\right]_i = \lim_{\Delta x \to 0} \frac{\left(\frac{\partial u}{\partial x}\right)_{i+1/2} + \left(\frac{\partial u}{\partial x}\right)_{i-1/2}}{\Delta x} + O(\Delta x^2) \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta x)^2} \qquad (3.8)$$

## 3.4 Metodo delle differenze finite applicato all'equazione di Poisson

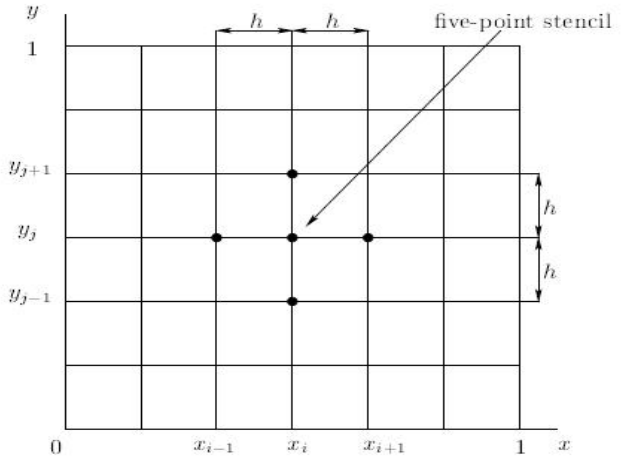
Una volta affrontato il problema generale del metodo delle differenze finite, può essere interessate andare ad applicarlo direttamente sulle equazioni di nostro interesse. Nella fattispecie, lo applicheremo alla sola equazione di Poisson, che come detto precedentemente, utilizzeremo per ricavare i termini $z_n$. Esistono due tipi di applicazioni possibili: il caso mono-dimensionale e il caso bidimensionale. Noi svilupperemo il caso bidimensionale, che è un estensione del caso mono-dimensionale, con la differenza che $u$ non dipende solo dalla variabile x, ma anche dalla variabile y, il che implica calcolare due derivate parziali e quindi applicare il metodo delle differenze finite sia rispetto a x sia rispetto a y. Per far ciò, si faccia dunque riferimento al seguente problema ai valori iniziali:

$$\begin{cases} \dfrac{\partial^2}{\partial x^2}u(x,y) + \dfrac{\partial^2}{\partial y^2}u(x,y) = f(x,y) \\ u(0,y) = u(\alpha,y) = u(x,0) = u(x,\alpha) = 0 \qquad \forall (x,y) \in [0,\alpha] \end{cases} \qquad (3.9)$$

in cui il dominio considerato è un quadrato con lato di ampiezza pari ad $\alpha$. Dal momento che i nostri domini non sono continui ma discreti e regolari, un immagine, mediante uso della (3.7) possiamo riscrivere (3.9) in forma discretizzata come:

$$\begin{cases} \dfrac{\partial^2}{\partial {x_i}^2} u_{i,j} + \dfrac{\partial^2}{\partial {y_j}^2} u_{i,j} = f_{i,j} \\ u_{0,j} = u_{1,j} = u_{i,0} = u_{i,1} \qquad\qquad \forall (i,j) \in [0,1] \end{cases} \tag{3.10}$$

Le derivate seconde in questo caso vanno intese in termini di approssimazione in serie del limite del rapporto incrementale. Nel seguente caso il dominio di applicazione è rappresentato da una griglia che arbitrariamente abbiamo definito come: $D\{x \in [0,1] \cup y \in [0,1]\}$ , tale scelta di fatto si avvicina molto all'idea di un immagine, e quindi di un dominio campionato. Inoltre assumiamo che $f$ esista e sia continua. Di seguito inoltre riportiamo una rappresentazione grafica di $D$ ,suddiviso per dicotomia, in tanti quadrati di dimensione $h \times h$ che costituiscono una griglia. Tale rappresentazione ci consente di visualizzare, in maniera rapida, i punti di interesse per l'applicazione del metodo.




A questo punto facendo riferimento alla (3.8) e alla figura, possiamo scrivere le seguenti espressioni:

$$\frac{\partial^2}{\partial {x_i}^2} u_{i,j} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} \tag{3.11}$$

$$\frac{\partial^2}{\partial {y_j}^2} u_{i,j} = \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2} \tag{3.12}$$

la (3.11) e la (3.12) rappresentano le derivate seconde parziali rispetto a x e a y (con approssimazione $O(h^2)$ che compaiono in (3.10) e (3.9). Se le combiniamo otteniamo un espressione del tipo:

$$\frac{\partial^2}{\partial {x_i}^2} u_{i,j} + \frac{\partial^2}{\partial {y_j}^2} u_{i,j} = \frac{u_{i+1,j} + u_{i-1,j} - 4u_{i,j} + u_{i,j+1} + u_{i,j-1}}{h^2} \tag{3.13}$$

e finalmente sostituendo la (3.13) in (3.10) otteniamo la formulazione del problema di Poisson in forma approssimata, ottenuta per applicazione del metodo delle differenze finite:

$$\begin{cases} \dfrac{u_{i+1,j} + u_{i-1,j} - 4u_{i,j} + u_{i,j+1} + u_{i,j-1}}{h^2} = f_{i,j} & \forall i,j = 1,\dots,N-1 \\ u_{i,0} = u_{i,N} = u_{0,j} = u_{N,j} = 0 & \forall i,j = 0,1,\dots,N \end{cases} \quad (3.14)$$

ma (3.14) rappresenta ha tutti gli effetti un sistema lineare del tipo $A * u = \boldsymbol{f}$ in cui A è una matrice di dimensioni $(N-1)^2 \times (N-1)^2$ ed $u\ e\ f$ sono dei vettori, di pari dimensione $(N-1)^2$. Nota $A\ e\ f$ possiamo dunque risolvere il sistema e ricavare $u$, l'incognita del problema di partenza (nel nostro caso i $z_n$). Tale formulazione ci consente quindi di implementare in maniera più o meno agevole, a seconda dei casi, algoritmi in grado di risolvere equazioni differenziali, e ciò viene fatto a partire dalla formulazione del metodo alle differenze finite, costruendo un opportuna matrice sparsa $A$ in forma di Toeplitz, di cui daremo una rappresentazione più avanti. Non solo ma questo ci consente di risolvere tali problemi in maniera efficiente e rapida, soprattutto se pl'implementazione di tali algoritmi viene fatta mediante ambienti di sviluppo interpreti quali: Matlab, Octave, che sono fortemente ottimizzati per il calcolo matriciale.

## 3.5 Struttura del sistema lineare

In questo paragrafo riprenderemo il discorso lasciato in sospeso alla fine del paragrafo precedente. Ovvero abbiamo visto che è possibile ricavare da (3.14) una matrice che adesso ci accingeremo a descrivere. Per fare questo riscriviamo la nostra equazione di Poisson in termini di $z$:

$$\frac{\partial^2}{\partial {x_i}^2} z_{i,j} + \frac{\partial^2}{\partial {y_j}^2} z_{i,j} = \rho_{i,j} \quad (3.15)$$

Dove abbiamo già scritto la nostra equazione riferendoci a un dominio regolare e $\rho_{i,j}$ rappresenta il termine noto che descriveremo nel paragrafo successivo. Possiamo dunque applicare direttamente il metodo delle differenze finite ottenendo quindi un'espressione del tutto analoga (3.14) che per completezza riscriviamo con il cambio di notazione:

$$\begin{cases} \dfrac{z_{i+1,j} + z_{i-1,j} - 4z_{i,j} + z_{i,j+1} + z_{i,j-1}}{h^2} = \rho_{i,j} & \forall i,j = 1,\dots,N-1 \\ z_{i,0} = z_{i,N} = z_{0,j} = z_{N,j} = 0 & \forall i,j = 0,1,\dots,N \end{cases} \quad (3.16)$$

Anche in questo caso ovviamente abbiamo ricavato tale espressione facendo riferimento a una griglia suddivisa in quadrati di egual area pari proprio ad $h^2$. Ma (3.16) può essere riscritta anche in termini matriciali come:

$$\boldsymbol{A}z = \rho \quad (3.17)$$

Dove $A$ è la matrice di Poisson e presenta una struttura a blocchi di tipo BTTB (block Toeplitz with Toeplitz blocks), dove B sono delle matrici diagonali di termini unitari mentre T ha come elementi in diagonale -4, e le diagonali inferiori e superiori anch'esse unitarie. Se dunque per semplicità ipotizziamo di avere una griglia quadrata di dimensione quattro ($N = 4$), che rappresenta la nostra

immagine, la matrice di Poisson risultante sarà pertanto di dimensione pari a nove ($(N-1)^2 \times (N-1)^2$), quindi sarà così strutturata:

$$A = \begin{pmatrix} T & B & 0 \\ B & T & B \\ 0 & B & T \end{pmatrix}$$

$$T = \begin{pmatrix} -4 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & -4 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

### 3.5.2 Proprietà della matrice del sistema

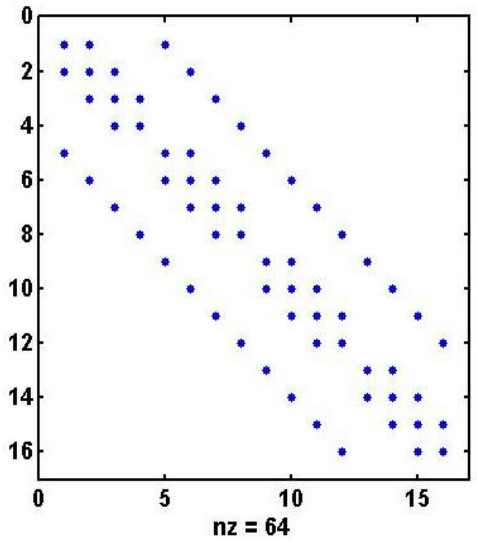
In questo sottoparagrafo verranno descritte tutte le proprietà della matrice di Poisson che utilizzeremo per la scelta dei metodi risolutivi più consoni. Come prima proprietà, dall'esempio precedente si evince subito che la matrice presenta una struttura a blocchi. In realtà $A$, oltre ad essere una matrice a blocchi, più nello specifico è di tipo: *Toeplitz.* Infatti una matrice $T$ si definisce di *Toeplitz* se gli elementi lungo una diagonale sono costanti, il tutto si traduce come:

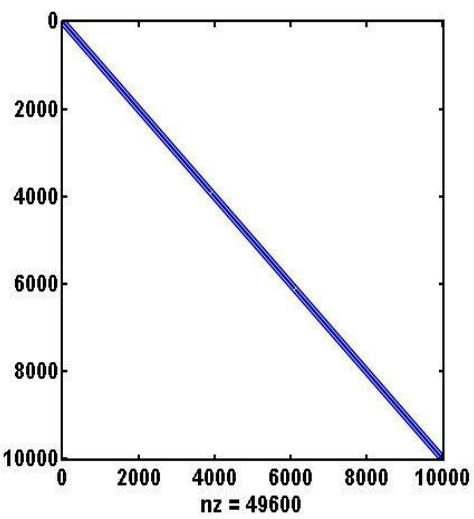
$$T_{i,j} = t_{i-j} \qquad i,j = 1, \dots, N \qquad (3.18)$$

Un altro modo per dimostrare tale proprietà consiste nel vedere $A$ come una matrice multi-indice. Infatti possiamo gestire un indice per la localizzazione di un blocco, e un altro per localizzare gli elementi all'interno del medesimo. In altri termini una matrice multi - indice è definita di Toeplitz se si verifica la seguente proprietà:

$$T_{\binom{i_1}{m}\binom{j_1}{n}} \quad dipende\ i_1 - j_1 \quad \text{e} \quad T_{\binom{M}{i_2}\binom{N}{j_2}} \quad dipende\ i_2 - j_2 \qquad (3.19)$$

Dome $m, n$ identificano l'elemento i-esimo all'interno del blocco $M, N$. Per tutte le matrici caratterizzate da tale proprietà esistono tutta una serie di teoremi che ci consentono di utilizzare metodi di risoluzione di sistemi lineari particolari che consentono di eseguire il prodotto matrice termine noto, in modo rapido facendo uso della fft, avremo modo più avanti di approfondire tale argomento. Oltre ad essere una matrice di Toeplitz, $A$ è inoltre una matrice a *banda*, il che significa che è possibile individuare una zona centrale dove gli elementi assumono valori diversi da zero mentre tutte le altre componenti sono nulle:



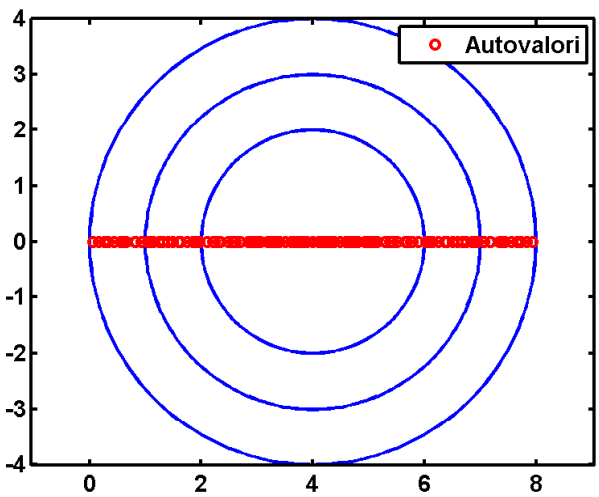



L'ampiezza della banda è evidentemente variabile al crescere delle dimensioni della griglia. Come per le matrici Toeplitz, anche per le matrici a banda esistono una serie di metodi risolutivi standard altamente ottimizzati. Tutto questo per convincervi del fatto che il modello differenziale non è stato poi scelto così a caso. Tutte queste proprietà infatti ci consentiranno di ottimizzare il processo di integrazione, il che ci assicurerà prestazioni di calcolo altamente performanti.

Fin qui ci siamo limitati a descrivere delle proprietà prettamente strutturali, ma la nostra matrice possiede tutta un'altra serie di proprietà algebriche importanti. Prima di tutto è *simmetrica,* per rendersene conto è sufficiente calcolare la differenza: $A - A^T$(o alternativamente $A^T - A$). Tale differenza ha come risultato una matrice di zeri, in altri termini, la trasposta coincide con la matrice di partenza. Questo implica avere autovalori tutti reali. Non solo, è anche *definita positiva,* il che vuol dire che tutti gli autovalori sono strettamente maggiori di zero, il che implica necessariamente la proprietà di non *singolarità*. Per dimostrare la proprietà di definizione positiva, si può fare riferimento ai teoremi di Gershgorin. Quest'ultimi sono molto importanti e trovano una miriade di applicazioni in ambito ingegneristico, come ad esempio nel controllo della stabilità dei sistemi dinamici. Infatti nelle applicazioni non si è sempre interessati a conoscere la posizione precisa di ogni singolo autovalore (anche perché in certi casi non è possibile), piuttosto siamo interessati a sapere in linea di massima in quale emisfero del piano di Gauss sussistono. I teoremi di Gershgorin ci consentono quindi di fare fronte a tale prerogativa. Il primo teorema afferma che se è assegnata una matrice quadrata $A$, si definiscono i cerchi riga gli insiemi:

$$R_i = \left\{ z \in \mathbb{C} : |z - a_{ii}| \leq \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| \right\}$$

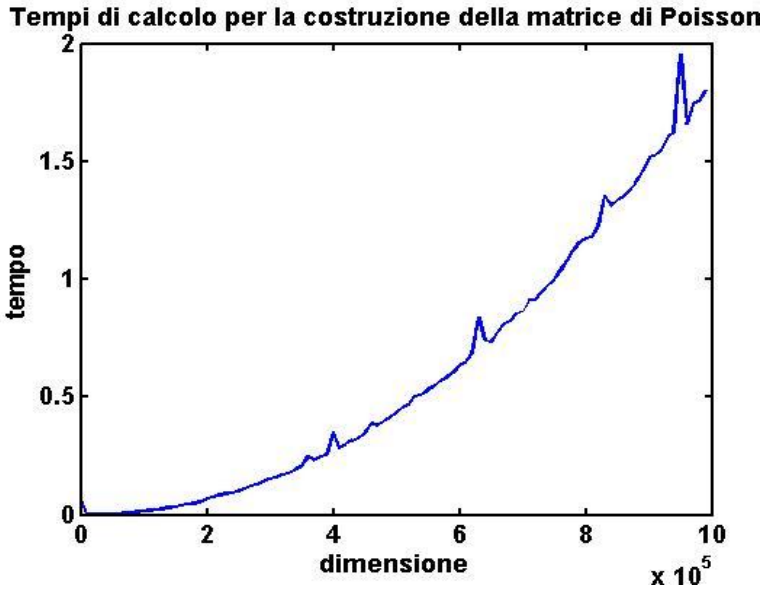
allora si può dimostrare che tutti gli autovalori di A sono contenuti nell'unione dei cerchi riga. Esiste inoltre un teorema analogo per i cerchi colonna che però noi non daremo. Nel caso della matrice di Poisson abbiamo a che fare con delle circonferenze riga tutte concentriche e di centro quattro, che corrisponde al valore assunto dai termini della diagonale del blocco $D$. Se per esempio la nostra matrice di Poisson, fosse di dimensione $256 \times 256$, otterremmo una rappresentazione dei cerchi di Gershgorin così fatta:



La circonferenza più esterna ha un diametro all'incirca pari a otto, e come potete constatare vale la definizione precedente in quanto tutti gli autovalori sono contenuti all'interno della circonferenza con raggio maggiore, e quest'ultimi sono tutti strettamente positivi. Il che allo stesso tempo dimostra la proprietà di definizione positiva e non singolarità.

### 3.5.3 Costruzione della matrice del sistema

Rimane solo un punto a questo punto da chiarire, ovvero il come viene memorizzata la matrice $A$. Quest'ultima infatti nelle applicazioni può raggiungere dimensioni piuttosto importanti, e ciò ovviamente è in funzione della grandezza delle immagini che compongono i nostri data set. In realtà abbiamo evidenziato precedentemente che la maggior parte delle componenti di una matrice a banda sono degli elementi nulli che non è necessario memorizzare. Pertanto nella fattispecie $A$ può essere vista come una matrice sparsa, in cui vengono salvati solo gli elementi non nulli e la loro posizione all'interno della matrice stessa. Ciò consente evidentemente di risparmiare grosse quantità di memoria. Volendo fare un esempio, ipotizzando di voler costruire una matrice $A$ di dimensione $10000 \times 10000$, se quest'ultima venisse memorizzata per intero occuperebbe uno spazio di memoria pari a: 8 gigabytes, mentre la matrice memorizzata come matrice sparsa occuperebbe: 873608 bytes. In altri termini la matrice piena occupa uno spazio di memoria che è all'incirca cento volte più grande rispetto a quello occupato dalla sparsa. Un altro aspetto importante riguarda poi sicuramente la velocità con cui si costruisce tale matrice. Ovviamente la cosa migliore da fare sarebbe quella di utilizzare linguaggi di programmazione come C, che ci consentirebbero prestazioni di buon livello. Come però avrete capito leggendo il testo, per le nostre prove e per lo sviluppo dei vari algoritmi abbiamo fatto uso dell'ambiente di sviluppo Matlab, e pensare di costruire una matrice di questo tipo utilizzando uno script risulta veramente proibitivo. Fortunatamente lo stesso Matlab mette a disposizione una routine interna "*gallery*" in cui sono contenute tutte le matrici di interesse ingegneristico e di maggior utilizzo in campo matematico tra cui anche la nostra matrice di Poisson. Tali matrici vengono costruite in modo molto rapido e soprattutto vengono memorizzate in modo sparso, limitando quindi l'occupazione di memoria. Questo rappresenta pertanto un'altro tassello importante a favore del modello differenziale scelto.



## 3.6 Costruzione del termine noto

Una volta quindi chiarite tutte le proprietà della matrice di Poisson possiamo finalmente passare alla costruzione del termine noto. Evidentemente quest'ultimo sarà costruito a partire dalla conoscenza dei gradienti che abbiamo discusso nella capitolo due di questo lavoro. Conviene pertanto riscrivere la nostra equazione di Poisson come:

$$\frac{\partial}{\partial x}\left(\frac{\partial z(x,y)}{\partial x}\right)+\frac{\partial}{\partial y}\left(\frac{\partial z(x,y)}{\partial y}\right)=\rho \qquad (3.19)$$

Dove non abbiamo fatto altro che esplicitare le derivate seconde come il prodotto delle due derivate prime rispettivamente per la variabile x e y. A questo punto se si ricorda la definizione di gradienti data in (2.3) quest'ultimi non sono altro che le derivate parziali della nostra $z$ che per completezza riscriviamo:

$$\begin{cases}\dfrac{\partial z(x,y)}{\partial x}=p\\ \dfrac{\partial z(x,y)}{\partial y}=q\end{cases} \qquad (3.20)$$

sostituendo quindi le (3.20) in (3.19) ciò che si ottiene è un espressione del tipo.

$$\frac{\partial}{\partial x}p+\frac{\partial}{\partial y}q=\rho \qquad (3.21)$$

I nostri gradienti, come più volte sottolineato però non sono delle funzioni continue pertanto la (3.21) deve essere riscritta in forma discretizzata e quindi risulta:

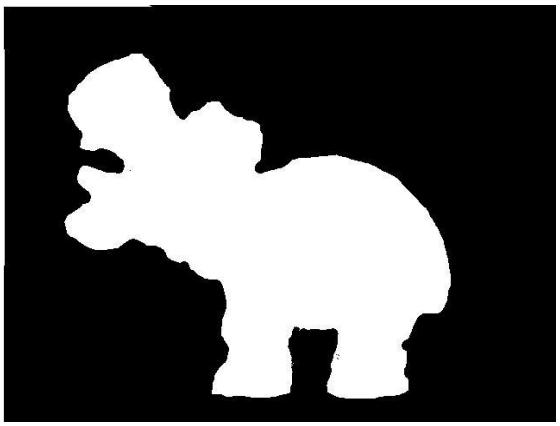
$$\begin{cases}\dfrac{\partial}{\partial x_i}p_{i,j}+\dfrac{\partial}{\partial y_j}q_{i,j}=\rho_{i,j}\\ \quad i,j=0,\dots,N-1\end{cases} \qquad (3.22)$$

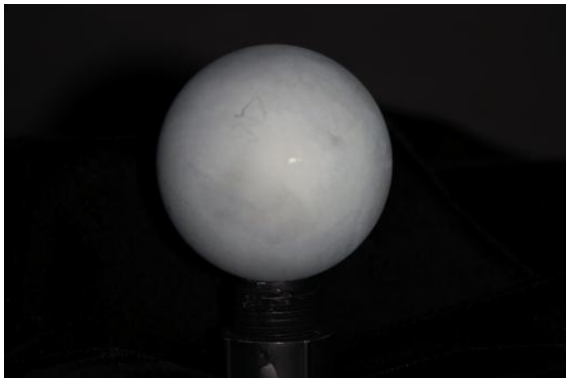
A questo punto possiamo applicare direttamente il metodo delle differenze finite in (3.22), questa volta però facendo riferimento a una griglia di dimensione $m\times n$ centrata in zero di dominio: $(-m,m)\cup(-n,n)$. L'espressione finale che si ottiene sarà pertanto:

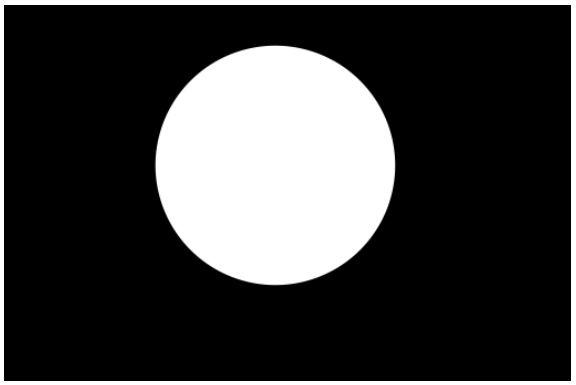
$$\begin{cases}\dfrac{p_{i+1,j}-p_{i,j}}{h}+\dfrac{q_{i,j+1}-q_{i,j}}{h}=\rho_{i,j} & i,j=1,\dots,m-1\,,n-1\\ \dfrac{p_{i+1,j}-p_{i,j}}{h}+\dfrac{q_{i,j+1}-q_{i,j}}{h}+z_0=\rho_{i,j} & i,j=0,m,n,-m,-n\end{cases} \qquad (3.23)$$

Dove in realtà la (3.23) non è un unica espressione in quanto dobbiamo tenere conto del dominio. Infatti se ci troviamo sulla frontiera dobbiamo provvedere a inserire un opportuna condizione al contorno $z_0$ che renda determinato il problema. Ovviamente se imponiamo che la condizione al contorno sia nulla su tutta la cornice è chiaro che la (3.23) si riduce ad un unica espressione. In realtà nelle applicazioni può capitare di non essere interessati a costruire il termine noto su tutto il

gradiente ma su una zona definita. Questo perché, in alcuni casi, gli oggetti che si vogliono ricostruire si trovano immersi all'interno di uno scenario che non ha alcuna importanza nella ricostruzione, anzi il più delle volte e causa di rumore e distorsione che quindi deve essere eliminato. Il problema può essere risolto in maniera molto semplice facendo uso di opportune maschere. Tali maschere hanno la stessa dimensione delle immagini (e quindi dei gradienti risultanti) che costituiscono i data set e possono essere viste come delle matrici logiche ovvero costituite solamente da termini unitari o nulli. Di seguito riportiamo per chiarezza alcuni esempi visivi:







La maschera ovviamente va utilizzata in fase di costruzione del termine noto, pertanto verranno considerati solo i punti dei gradienti i quali nella corrispondente maschera assumono un valore unitario (bianco). Se quest'ultimi cadono all'esterno viceversa, indipendentemente dal valore assunto in quel punto dal gradiente il termine noto varrà zero in quanto la maschera nel medesimo varrà zero (nero). Per rendere più chiaro il processo riportiamo uno pseudo codice della costruzione del termine noto:

```
[m n]=size(p,q),
[m1 n1]=size(mask),
check=check [m1=m AND n1=n]
if check==1
   for i=1:m-1
       for j=1:n-1
```

```
        ro(i,j)=p(i+1,j)-p(i,j)+q(i,j+1)-q(i,j),
      end
    end
else
error('mask and gradients must have the same dimension')
end
```

Di conseguenza come per la matrice $A$ anche il termine noto può essere visto come un vettore sparso, così facendo siamo ulteriormente in grado di limitare lo spazio occupato in memoria, non solo, ma allo stesso tempo riduciamo anche i tempi di calcolo riducendo il numero delle variabili.

## 3.7 Introduzione ai vari metodi di integrazione

Una volta quindi descritti gli elementi principali che entrano in gioco nel modello di integrazione, possiamo passare alla descrizione delle struttura dell'algoritmo. Come prima cosa si costruisce il termine noto che abbiamo descritto nel paragrafo precedente. A seconda del metodo di risoluzione utilizzato, quest'ultimo dovrà essere vettorializzato, il che comporta trasformare il gradiente immagine in un vettore colonna. Alternativamente continuerà ad essere un immagine. La fase immediatamente successiva consiste nella costruzione della matrice di Poisson a partire dalla conoscenza della dimensione del termine noto. In realtà questa operazione, come detto in precedenza, viene eseguita direttamente dalla routine di Matlab, per cui non ci preoccuperemo di tale fase. Come ultimo passaggio quindi non resta altro che risolvere il sistema lineare risultante che assumerà la forma (3.17). A tal proposito l'algebra lineare mette a disposizione un infinità di possibilità. Non sarà obbiettivo di questa tesi dare una panoramica completa di tutti i metodi risolutivi, (per una panoramica completa rimandiamo a [1] e [2]), piuttosto ci concentreremo su quelli più indicati per il nostro sistema lineare. Infatti dal punto di vista matematico non siamo in grado di poter affermare che esista a priori un metodo risolutivo migliore di tutti gli altri. Ogni approccio risolutivo va pertanto contestualizzato in base al modello matematico di rifermento che ci si ritrova ad affrontare. Il nostro lavoro quindi è stato quello di comprendere e analizzare le strutture del nostro modello cercando di sfruttarle ai fini di poter scegliere un metodo risolutivo consono. Di seguito pertanto riporteremo i modelli che rispondono a tale priorità. Questi ultimi verranno analizzati e confrontati ponendo in risalto eventuali punti di forza e criticità.

## 3.8 Risoluzione del sistema di Poisson con metodo diretto (Gauss)

Come è emerso dalla fine del paragrafo 3.7.3, in relazione alla dimensione delle immagini con cui ci ritroviamo a lavorare, la matrice di Poisson può raggiungere dimensioni davvero molto importanti. Saremmo pertanto giustificati nel pensare che l'impiego di un metodo diretto per la risoluzione del sistema sia una scelta del tutto azzardata. In realtà però la matrice $A$ abbiamo visto che ha una serie di proprietà importanti, tra le principali il fatto di essere una matrice a banda. In realtà sarebbe più

corretto parlare di matrice a "banda variabile", in quanto le diagonali unitarie tendono a spostarsi verso all'esterno al crescere di $A$, però di fatto la larghezza di banda poi si mantiene sempre molto stretta in relazione alle dimensioni della matrice. Tutto ciò sarebbe già sufficiente per giustificare la scelta di utilizzare un metodo risolutivo diretto, come ad esempio Gauss. Immaginiamo pertanto di voler risolvere un sistema lineare del tipo:

$$A\boldsymbol{x} = \boldsymbol{b} \qquad (3.24)$$

Utilizzare il metodo di Gauss comporta trasformare il sistema in un altro sistema triangolare equivalente. Questa operazione viene effettuata sostanzialmente sfruttando i tre principi di equivalenza che ci consentono di moltiplicare ciascuna riga per uno scalare, scambiare due o più righe del sistema, effettuare operazione di somma tra le righe. In conclusione, una volta ottenuto il sistema triangolare, quest'ultimo può essere risolto in modo molto rapido procedendo mediante: *back forward substitution* (all'indietro). A livello implementativo per poter fare ciò, abbiamo la necessità di gestire tre cicli for separati, ipotizzando poi che $A$ sia di dimensione $n \times m$, l'algoritmo risultante sarà così strutturato:

```
for k=1:n-1
  for i=k+1:n
   m_{i,k} = a_{ik} / a_{kk}
   for j=k+1:m
    a_{i,j}^{(k+1)} = a_{i,j}^{k} - m_{ik} a_{k,j}^{k}
    a_{i,k}^{(k+1)} = 0
    b_{i}^{(k+1)} =   b_{i}^{k} -  m_{i,k} b_{k}^{k}

   end
   end
end
```

Con il primo ciclo gestiamo il passo,con gli altri due invece, ci muoviamo all'interno della matrice ed effettuiamo operazioni di equivalenza per trasformare il sistema in una forma triangolare. Tale algoritmo è applicabile solamente se tutti i termini *pivot* $a_{kk}$ sono identicamente non nulli. Il che non è sempre verificato, motivo per il quale l'algoritmo più generale prevede un'opportuna operazione di controllo la quale verifica che l'elemento pivot sia non nullo, altrimenti quest'ultimo deve essere spostato semplicemente scambiando le righe del sistema in accordo con i principi di equivalenza. Però nel nostro caso la matrice di Poisson è *diagonalmente dominante* il che significa:

$$|\boldsymbol{a_{i,i}}| \geq \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{i,j}| \qquad i = 1, \dots, n$$

Pertanto i termini *pivot* sono sempre non nulli. Così facendo saremmo già in grado di risolvere il sistema, in realtà con l'algoritmo appena descritto siamo in grado di generare una fattorizzazione tale per cui possiamo riscrivere $A$ come:

$$A = LU \qquad (3.25)$$

dove $L$ e $U$ sono rispettivamente delle matrici triangolare inferiore e superiore così strutturate:

$$l_{i,j} = \begin{cases} m_{i,j} & i > j \\ 1 & i = 0 \\ 0 & i < j \end{cases} \qquad u_{i,j} = \begin{cases} a_{i,j} & i \leq j \\ 0 & i > j \end{cases}$$

Una volta costruite le matrici sarà nota la fattorizzazione e quindi possiamo risolvere il sistema (3.24) come:

$$\begin{cases} L\boldsymbol{y} = \boldsymbol{b} \\ U\boldsymbol{x} = \boldsymbol{y} \end{cases} \qquad (3.26)$$

In realtà l'utilizzo della fattorizzazione $LU$ va ben oltre la semplice risoluzione del sistema. Infatti può essere anche utilizzata per calcolare il determinate oppure per calcolare l'inversa di $A$, con complessità computazionale decisamente inferiore rispetto ai metodi standard [2].

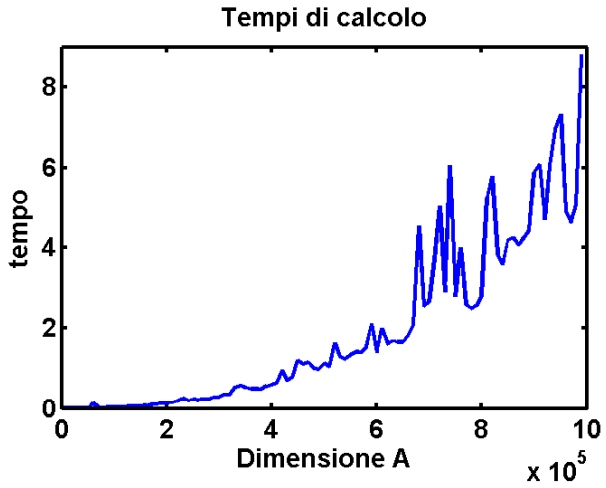
Tornando al nostro sistema di partenza quindi, applicando Gauss abbiamo una complessità che è pari a: $\boldsymbol{O}(m^3n^3)$, dove m ed n sono le dimensioni dell'immagine in pixel. Questa poi nella pratica diventa $\boldsymbol{O}(\frac{m^3}{3}\frac{n^3}{3})$ il che indica un carico computazionale piuttosto importante che è tipico dei metodi diretti. Ma la matrice di Poisson, come detto più volte, è una matrice a banda variabile che può essere stimata, e in tal caso non è necessario applicare il metodo su tutta la matrice, ma limitarlo alla sola banda il che riduce enormemente la complessità. In altri termini se l'ampiezza della banda è pari $r$ l'algoritmo diventa:

```
for k=1:n-1
  for i=k+1: k+r
   m_{i,k} = a_{ik} / a_{kk}
   for j=k+1:k+r
    a_{i,j}^{(k+1)} = a_{i,j}^{k} − m_{ik} a_{k,j}^{k}
    a_{i,k}^{(k+1)} = 0
    b_i^{(k+1)} =  b_i^{k} −  m_{i,k} b_k^{k}
     end
    end
end
```

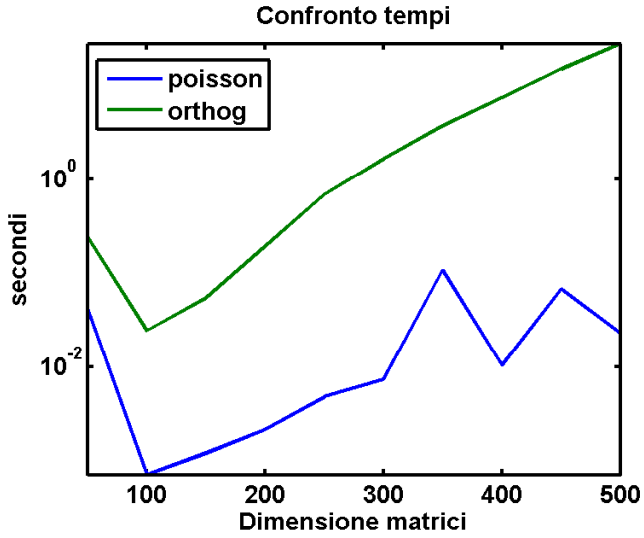
Ciò comporta in termini di complessità $\boldsymbol{O}(m\,n^3)$ o analogamente $\boldsymbol{O}(m^3n)$. Se $m = n$, (come nel nostro caso, dato che $A$, per come è costruita può essere solo quadrata) la complessità sarà pari a: $\boldsymbol{O}(n^4)$. Contrariamente se $A$ fosse piena avremmo: $\boldsymbol{O}(n^6)$, due ordini di grandezza sotto rispetto al caso standard che può fare la differenza se i sistemi di dimensione importanti.

Tutto questo a livello implementativo in Matlab può essere fatto semplicemente utilizzando il comando `A\b`. Infatti quest'ultimo è in grado di riconoscere la struttura a banda della matrice di Poisson e applica l'algoritmo appena descritto. Pensare di implementare l'algoritmo in prima persona non appare un'operazione conveniente a meno che non si utilizzino linguaggi di programmazione come il C. Come detto più volte Matlab fa uso di routine interne ottimizzate per lo più scritte in C e Fortran, ma a livello di esecuzione, i cicli delle routine interne sono tremendamente più rapide rispetto a quelli esterni gestiti dagli utenti.

Di seguito riportiamo alcuni test effettuati sul metodo di integrazione appena descritto. Come prima cosa abbiamo risolto il sistema costruendo la matrice di Poisson e dei termini noti casuali e abbiamo misurato i tempi di calcolo al crescere della matrice.

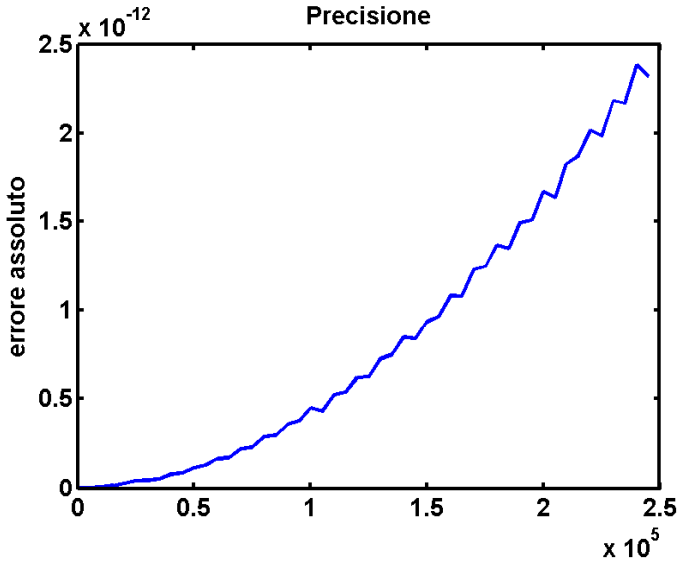



Il calcolatore utilizzato per il test è il già citato intel i-core 5 con sistema operativo Mac-Os. Come si può constatare dalla scala dei tempi, a meno di qualche picco, l'andamento è tendenzialmente regolare e per risolvere un sistema caratterizzato da un milione di variabili, non si impiegano più di otto secondi. Il che per un metodo diretto è veramente sorprendente. Ovviamente poi ci siamo preoccupati di eseguire un ulteriore confronto adottando una matrice piena e risolvendo il sistema utilizzando lo stesso termine noto. La matrice di confronto anche in questo caso è stata prelevata dalla routine gallery ed abbiamo optato per la *orthog* . Come è intuibile dal nome la matrice che viene costruita è una base ortogonale che ha la caratteristica di avere un condizionamento unitario per qualsiasi dimensione e ciò consente di ottenere un confronto più affidabile.



Come si può osservare dalla figura le scale dei tempi sono totalmente differenti. Nel caso della matrice piena per risolvere un sistema di dimensione dieci mila, pertanto abbastanza piccolo, si superano abbondantemente i trenta secondi. La motivazione risiede nel fatto che in questo caso si utilizza l'algoritmo di Gauss standard e pertanto il processo di triangolarizzazione sulla matrice piena richiede un costo computazionale molto elevato. Mentre nel secondo caso, (matrice di Poisson), Matlab ha riconosciuto la matrice a banda e quindi applica il metodo risolutivo ottimizzato.

L'ultimo test che abbiamo effettuato concerne la precisione del metodo. Per far ciò si è costruito un banco di prova imponendo una soluzione unitaria nota a priori e di conseguenza abbiamo ricavato il termine noto tale da offrirci la soluzione ricercata e risolto più volte il sistema al crescere della matrice e del termine noto. I risultati ottenuti sono riportati nella figura sottostante:




Il fatto di utilizzare un metodo diretto ci consente di ottenere delle soluzioni piuttosto precise e ciò nella pratica assicura ricostruzioni abbastanza fedeli in relazione alla qualità dei gradienti elaborati. La precisione tende a diminuire all'aumentare della matrice di Poisson, ma ovviamente ciò dipende dal condizionamento che incrementa al crescere della matrice. E' altresì vero che siamo su una scala di precisione di $10^{-12}$ e una scala di grandezza per la matrice pari a: $10^5$, ovvero abbastanza importante.

## 3.9 Introduzione ai metodi iterativi

Abbiamo visto come l'utilizzo del metodo di Gauss nelle matrici a banda di grosse dimensioni garantisca ottime performance sia in termini di tempo di calcolo che di precisione. Ci siamo poi immediatamente domandanti se fosse stato possibile migliorare utilizzando metodi iterativi. Infatti questi ultimi risultano convenienti soprattutto quando le matrici in questione sono strutturate o sparse, come la nostra. Differentemente a un metodo diretto, che ha l'effetto di alterare la matrice di partenza aggiungendo elementi nulli e quindi triangolarizzando, con un metodo iterativo ciò

non accade, la matrice rimane intatta. In altri termini, se si vuole risolvere il sistema $A\boldsymbol{x} = \boldsymbol{b}$, si genera a partire da un vettore iniziale $x^{(0)}$ una successione di vettori $x^{(k)}$ che convergono alla soluzione del problema. E' altresì vero che se in un metodo diretto l'unica fonte di errore è data dall'aritmetica finita di macchina, nel caso di metodi iterativi si aggiungono anche problemi di troncamento. Questi ultimi sono dovuti al fatto che il limite cercato deve essere approssimato troncando la successione dei termini $x^{(k)}$ per un indice sufficientemente grande. Ciò inoltre determina il problema di avere un criterio d'arresto. Come detto più volte non sarà obbiettivo di questo lavoro testare tutti i metodi iterativi, ci concentreremo in particolare sul *metodo del gradiente* [1]. Anche qui la scelta deriva dal fatto di sfruttare una caratteristica della matrice di Poisson, ovvero che è simmetrica ed è definita positiva.

## 3.10 Il metodo del gradiente

Se $A$ è una matrice simmetrica e definita positiva, possiamo considerare la forma quadratica:

$$\varphi(y) = \frac{1}{2} y^T Ay - y^T b \qquad (3.27)$$

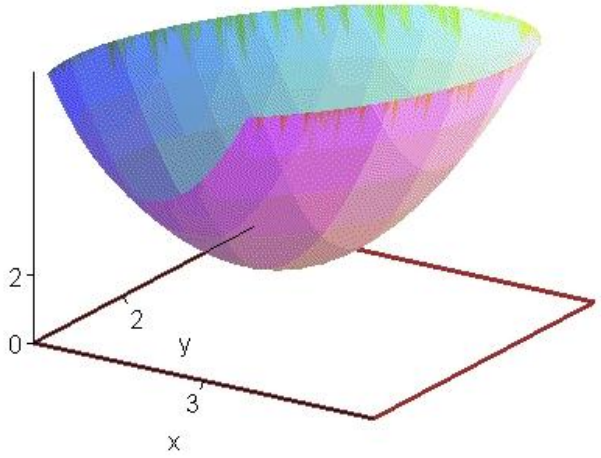
La funzione sarà minima nel punto in cui si annulla il suo gradiente e quindi:

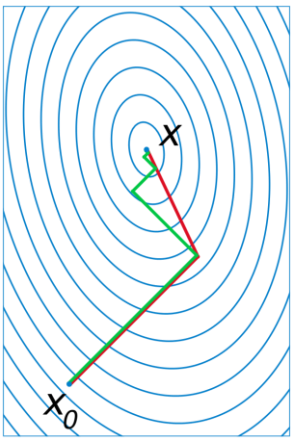
$$\nabla\varphi(y) = Ay - b = 0 \qquad (3.28)$$

In altri termini minimizzare la (3.27) significa risolvere un sistema lineare del tipo: $A\boldsymbol{x} = \boldsymbol{b}$**.** Per far ciò si può pensare dunque di minimizzare $\varphi(y)$ utilizzando un metodo iterativo non stazionario del primo ordine del tipo:

$$x^{(k+1)} = x^{(k)} + \alpha_k d^{(k)} \qquad (3.29)$$

Per far ciò abbiamo la necessità di inizializzare il metodo con un vettore iniziale $x^{(0)}$, lungo le direzioni di discesa $d^{(k)}$, con passi di lunghezza $\alpha_k$.



Indipendentemente dalla direzione di discesa, è possibile determinare il minimo $\varphi(x^{(k+1)})$ rispetto alla variabile $\alpha = \alpha_k$. Pertanto risulta:

$$\varphi\left(x^{(k+1)}\right) = \frac{1}{2}\left(x^{(k)} + \alpha d^{(k)}\right) A\left(x^{(k)} + \alpha d^{(k)}\right) - \left(x^{(k)} + \alpha d^{(k)}\right)^T b$$

che possiamo riscrivere in una forma più compatta come:

$$\varphi\left(x^{(k+1)}\right) = \varphi\left(x^{(k)}\right) + \frac{1}{2}(d^{(k)})^T\, Ad^{(k)}.\alpha^2 - (d^{(k)})^T r^{(k)}.\alpha = 0$$

derivando infine rispetto ad $\alpha$ segue:

$$\frac{d}{d\alpha}\varphi\left(x^{(k+1)}\right) = (d^{(k)})^T\, Ad^{(k)}.\alpha - (d^{(k)})^T r^{(k)} = 0 \qquad (3.30)$$

quindi in conclusione il minimo si ottiene dalla (3.30) semplicemente come rapporto:

$$\alpha_k = \frac{(d^{(k)})^T r^{(k)}}{(d^{(k)})^T\, Ad^{(k)}} \qquad (3.31)$$

Nel metodo del gradiente, si cerca di scegliere come direzione quella di massima discesa, che coincide con quella opposta alla direzione del gradiente nel punto $x^{(k)}$. Quindi nota la direzione del gradiente in automatico è nota la direzione di massima discesa. Quest'ultima inoltre coincide con il residuo al passo $k$, pertanto l'iterazione (3.29) deve essere modifica nella forma:

$$x^{(k+1)} = x^{(k)} + \alpha_k r^{(k)} \qquad (3.32)$$

In cui $r^{(k)}$ sta ad indicare il residuo al passo $k$. Inoltre per ridurre la complessità computazionale esiste un trucco che consente di aggiornare il residuo, senza doverlo ogni volta determinare a ogni iterazione:

$$r^{(k+1)} = r^{(k)} + \alpha_k A r^{(k)} \qquad (3.33)$$

Nella pratica tale metodo non viene mai utilizzato in questa forma in quanto la convergenza risulta molto lenta. Pertanto viene modificato introducendo una matrice *di precondizionamento* in sostituzione ad $A$, che tipicamente è più semplice da invertire e ciò consente una convergenza più rapida come verrà esplicitato meglio in seguito. In realtà esiste un ulteriore metodo, del tutto analogo al metodo del gradiente, che risulta preferito in quanto è in grado di stimare con più precisione le direzioni di massima discesa. Tale metodo è meglio noto come:*"metodo del gradiente coniugato",* che ci accingiamo a descrivere.

### 3.10.2 Metodo del gradiente coniugato

Come introdotto alla fine del paragrafo precedente, il metodo del gradiente coniugato è del tutto analogo al metodo del gradiente, ad eccezione della scelta della direzione di discesa che è più

accurata. Per altro se quest'ultimo viene correttamente precondizionato è in grado di convergere a una soluzione con un ridotto numero di iterazione ed una precisione piuttosto accurata.

Sia pertanto $p$ una generica direzione in un sottospazio vettoriale $V$, allora si può dimostrare che il vettore $x^{(k)}$ si definisce *ottimale* rispetto a $p$, se la direzione $p$ è ortogonale rispetto al residuo $r^{(k)}$. In altri termini il prodotto scalare deve essere identicamente nullo:

$$p^T r^{(k)} = 0 \qquad (3.34)$$

Nel metodo del gradiente ciò si verifica al primo passo di seguito alla scelta del parametro $\alpha_k$, di conseguenza il punto $x^{(k+1)}$è ottimale rispetto $r^{(k)}$. Ma ciò non è più vero per il vettore $x^{(k+2)}$. Pertanto con il metodo del gradiente coniugato si cerca di superare questo limite mantenendo la condizione di ottimalità a ogni singolo passo. Supponiamo dunque $x^{(k)}$ sia ottimale rispetto a $p$, e quindi vale (3.34), quindi al primo passo risulterà:

$$x^{(k+1)} = x^{(k)} + q$$

Affinché $x^{(k+1)}$ sia ottimale rispetto a p dovrà essere verificata la seguente condizione:

$$-p^T A\, q = 0 \qquad (3.35)$$

cioè le direzioni $p\ e\ q$ devono essere $A - ortogonali\ o\ A - coniugate$. Imponendo a questo punto $p^{(0)} = r^{(0)}$ si possono considerare le seguenti direzioni:

$$p^{(k+1)} = r^{(k)} - \beta_k p^k \qquad (3.36)$$

Ma $p^{(k+1)}$ e $p^k$ devono essere $o\ A - coniugate$, il che implica:

$$\left(p^{(k)}\right)^T A\, p^{(k+1)} = 0 \qquad (3.37)$$

La (3.37) si traduce in una condizione simile alla (3.31) che possiamo riscrivere come:

$$\beta_k = \frac{\left(p^{(k)}\right)^T A\, r^{(k+1)}}{(p^{(k)})^T A\, p^{(k)}} \qquad (3.38)$$

Così facendo si può dimostrare che $p^{(k+1)}$ è $A - coniugate$ con tutte le direzioni precedentemente generate. Per concludere riportiamo uno pseudo codice dell'algoritmo che raccoglie le operazioni appena descritte

***define*** $\boldsymbol{x^{(0)}}$
$\boldsymbol{r^{(0)} = b - A\, x^{(0)}}$
$\boldsymbol{p^{(0)} = r^{(0)}}$
$\boldsymbol{k = 0}$
***Do while***
$\boldsymbol{s = Ap^{(k)}}$

$\boldsymbol{\delta = \left(p^{(k)}\right)^T s}$
$\boldsymbol{\alpha_k = \left(p^{(k)}\right)^T r^{(k)} \delta^{-1}}$
$\boldsymbol{x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)}}$
$\boldsymbol{r^{(k+1)} = r^{(k)} + \alpha_k s}$
$\boldsymbol{\beta_k = s^T r^{(k+1)} \delta^{-1}}$
$\boldsymbol{p^{(k+1)} = r^{(k+1)} + \beta_k p^k}$
$\boldsymbol{k = k + 1}$
***Until*** *(convergenza)*

Per quanto l'algoritmo descritto rispetti la condizione di ottimalità, presenta problemi di lentezza e di convergenza. Questo problema può essere in parte risolto, come per il caso del metodo del gradiente, definendo un opportuno precondizionatore. Pertanto l'algoritmo appena descritto può essere modificato in tal senso inserendo una condizione iniziale di precondizionamento. Prima però di passare alla nuova descrizione preferiamo dedicare un piccolo sotto paragrafo all'analisi del precondizionatore.

### 3.10.3 Il precondizionamento

Come abbiamo detto precedentemente, un precondizionatore è una matrice che consente di accelerare la velocità di convergenza nei metodi iterativi. E' possibile limitare il numero delle iterazioni e quindi tempi di calcolo. Un metodo precondizionato può essere rappresentato nella forma:

$$x^{(k+1)} = Bx^{(k)} + f \qquad (3.39)$$

in cu $B = P^{-1}N = (I - P^{-1}A)$ e $f = P^{-1}b$. Per il principio della consistenza il che equivale a un sistema lineare che assume la seguente forma:

$$(I - B)x = f \qquad (3.40)$$

sostituendo la definizione di B data precedentemente si ottiene quindi:

$$P^{-1}Ax = P^{-1}b \qquad (3.41)$$

Dove $P$ rappresenta il precondizionatore. La scelta di tale parametro non è affatto banale e scontata e dipende fortemente dalle caratteristiche della matrice $A$. Quando quest'ultima pertanto assume dimensioni importanti si cerca di introdurre precondizionatori tali da preservare la sparsità e la struttura originale. In ogni caso, indipendentemente dal tipo di precondizionatore, quest'ultimo deve essere tale per cui: $P^{-1}A \cong I$, in cui $P$ risulta una matrice molto più semplice da invertire rispetto ad $A$

Una famiglia di ottimi precondizionatori per le matrici sparse, come nel nostro caso, è rappresentata dalle fattorizzazioni LU incomplete. Abbiamo visto nel paragrafo 3.8 che in generale una fattorizzazione LU presenta una complessità: $\boldsymbol{O}(n^3)$, quindi abbastanza elevata. Non solo ma nel caso di matrici sparse accade che i fattori triangolari presentino un numero di componenti decisamente superiore rispetto alla matrice $A$ che li ha generati. Il che implica occupazione di

memoria e tempi di calcolo ovviamente superiori. Per tutta questa serie di motivazioni si preferiscono le fattorizzazioni LU incomplete nelle quali vengono calcolate le sole componenti il cui valore assoluto supera una certa soglia prefissata $\sigma$. Il valore di $\sigma$ viene dunque scelto in modo tale che la fattorizzazione LU risultante approssimi con una buona precisione la matrice, in altri termini: $A \cong LU = P$.

Ma abbiamo dimostrato che la matrice di Poisson è simmetrica e definita positiva e in questo caso è possibile definire un'altra classe di precondizionatori che fanno uso della fattorizzazione di Cholesky incompleta. Così facendo otterremo una matrice triangolare superiore $R$ tale che $A \cong R^T R = P$, in cui $P$ rappresenta il nostro precondizionatore finale. A livello implementativo, per costruire un precondizionatore in Matlab, possiamo ricorrere alla routine *cholinc*, la chiamata standard pertanto risulta: `R=cholinc(A,sigma)`. In realtà nelle ultimi versioni di Matlab la routine è stata aggiornata con una nuova molto più efficiente chiamata: *ichol,* che sta appunto per incomplet Cholescky factorization. Tale routine a differenza della precedente mette a disposizione una serie di opzioni utili a massimizzare l'efficienza dei precondizionatori risultanti. Per la matrice sparse, come suggerito dagli stessi creatori, è consigliato l'utilizzo dell'opzione '*midchol*' ,la chiamata completa: `R=ichol(A,struct('midchol','on'))`. Abbiamo avuto modo di constatare che i precondizionatori risultanti effettivamente sono molto efficienti rispetto ad altri e riducono considerevolmente il numero delle iterazioni, avremo modo più avanti di mostrare qualche risultato.

### 3.10.4 Metodo del gradiente coniugato precondizionato

Una volta quindi definito il nostro precondizionatore quest'ultimo può essere utilizzato per aumentare la velocità del nostro metodo iterativo. Più in generale il metodo del gradiente coniugato precondizionato e spesso indicato con la sigla PCG e l'algoritmo risultante è del tutto analogo a quello mostrato in 3.10.2.

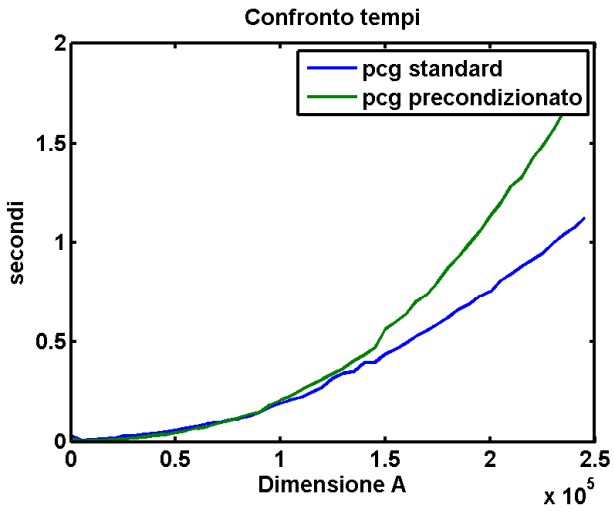
***define*** $\boldsymbol{x^{(0)}}$
$\boldsymbol{r^{(0)} = b - A\,x^{(0)}}$
***Solve*** $\boldsymbol{Pz^{(0)} = r^{(0)}}$
$\boldsymbol{p^{(0)} = r^{(0)}}$
$\boldsymbol{k = 0}$
***Do while***
$\boldsymbol{s = Ap^{(k)}}$
$\boldsymbol{\delta = \left(p^{(k)}\right)^T s}$
$\boldsymbol{\alpha_k = \left(p^{(k)}\right)^T r^{(k)} \delta^{-1}}$
$\boldsymbol{x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)}}$
$\boldsymbol{r^{(k+1)} = r^{(k)} + \alpha_k s}$
$\boldsymbol{\beta_k = s^T r^{(k+1)} \delta^{-1}}$
$\boldsymbol{p^{(k+1)} = r^{(k+1)} + \beta_k p^k}$
$\boldsymbol{k = k + 1}$
***Until*** *(convergenza)*

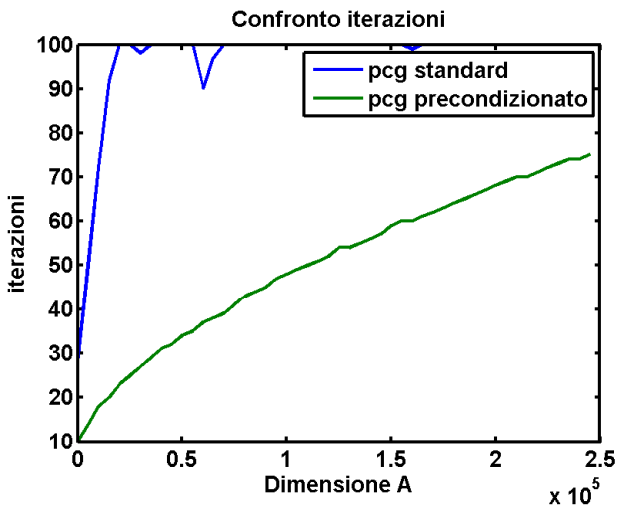
L'unica aggiunta rispetto a quello visto precedentemente è proprio data dalla presenza di un sistema iniziale $Pz^{(0)} = r^{(0)}$, che viene risolto molto velocemente se il precondizionatore è semplice da invertire.

Fortunatamente anche in questo caso Matlab mette a disposizione una function. La chiamata standard può essere formulata in questo modo: [x res ite]=pcg(A, ro, tol, maxit, R, R', x0), in cui A rappresenta la nostra matrice di Poisson e ro il termine noto costruito a partire dai gradienti. Tol e maxit invece rappresentano rispettivamente: la tolleranza sul residuo e il numero massimo di iterazioni, informazioni utili a gestire i criteri d'arresto. Successivamente possiamo inserire il nostro precondizionatore e un vettore iniziale che di default è un vettore iniziale che ha dimensione pari al termine noto. La funzione in output restituisce la soluzione, il valore del residuo e il numero di iterazioni eseguite, informazioni utili ai fini di test.

### 3.10.5 Test sul pcg e analisi comparata con il bcg

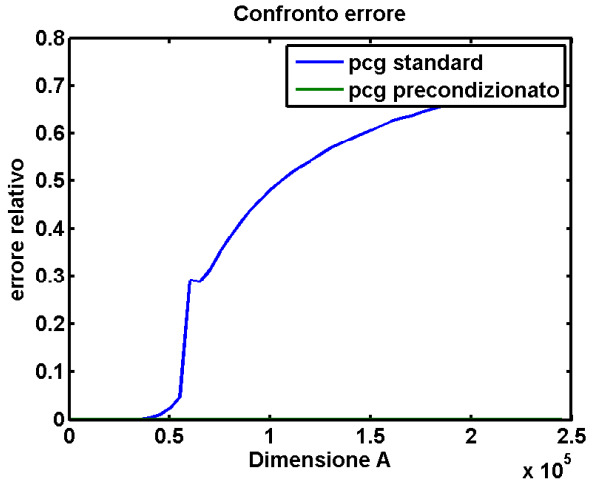
In questa sezione illustreremo alcuni risultati ottenuti utilizzando il metodo del gradiente. Alla fine del sotto paragrafo in cui parlavamo del precondizionatore vi avevamo promesso che vi avremmo mostrato dei risultati in merito all'effetto dell'utilizzo del precondizionatore. Come primo test costruiremo un banco di prova in cui come al solito utilizziamo la nostra matrice di Poisson e un termine noto puramente casuale e quindi risolveremo il sistema prima utilizzando il metodo del gradiente standard e successivamente il metodo precondizionato, salvando i tempi di calcolo e il numero delle iterazioni per valori crescenti di $A$. Di seguito riportiamo i risultati ottenuti utilizzando come valore di tolleranza e numero di iterazioni rispettivamente $10^{-6} e\ 100$ (valori default preimpostati da Matlab).

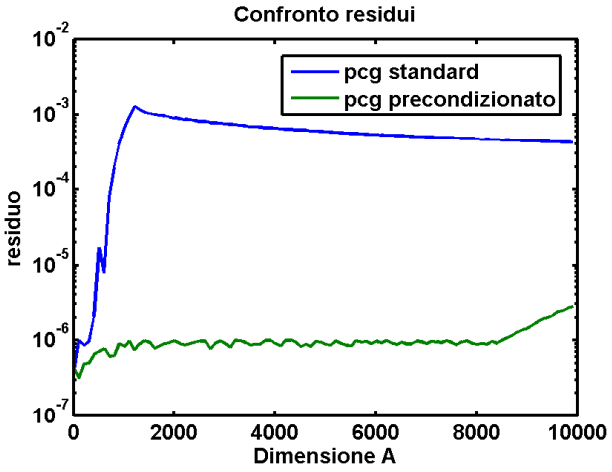



Nel caso del metodo non precondizionato è immediato osservare come il numero delle iterazioni arriva immediatamente a saturare anche con matrici non particolarmente grosse. Mentre se si utilizza il precondizionatore, non si raggiunge il valore massimo di iterazioni. A livello di tempi pare che non ci sia una grossa differenza, anzi addirittura il metodo standard sembrerebbe anche più rapido. Si deve però fare attenzione a una cosa, ovvero con il metodo senza precondizionatore

abbiamo evidenti problemi di convergenza, e questo è dimostrato dal numero di iterazioni che supera sempre il valore pre-impostato. Pertanto il test va completato confrontando i vari valori di residuo riscontrati e misurando l'errore relativo in norma di Frobenius a partire dalla conoscenza della soluzione vera e quella realmente calcolata dai due metodi. Come nel test precedente quindi ci siamo preoccupati di definire una soluzione random nota e conseguentemente abbiamo costruito il termine noto che abbiamo quindi utilizzato per la risoluzione dei vari sistemi.

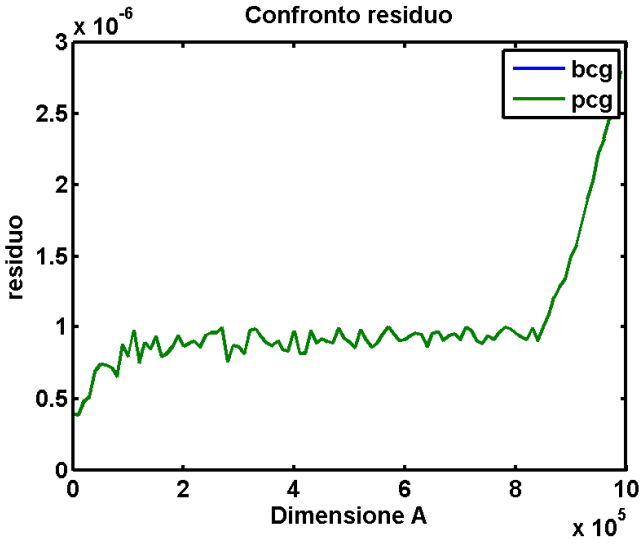


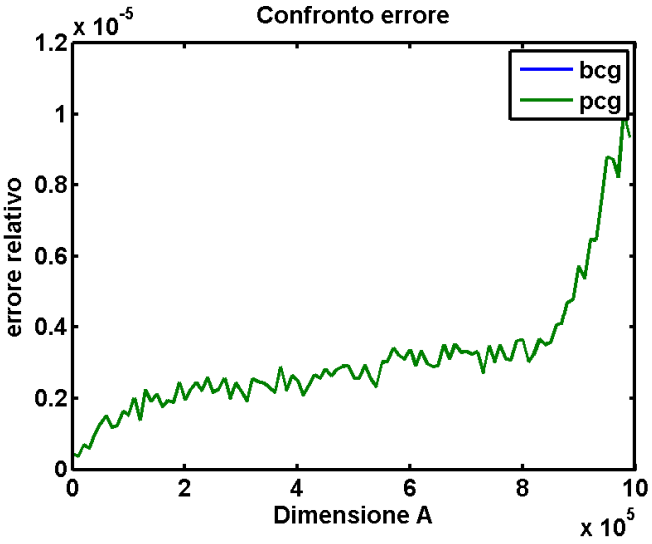




La soluzione calcolata con il pcg precondizionato è decisamente più precisa rispetta a quella calcolata senza e questo trend si evince anche dal confronto con i due residui. Pertanto il precondizionatore oltre a ridurre drasticamente il numero di iterazioni ci consente di ottenere delle soluzioni molto più precise. E la motivazione è abbastanza ovvia in quanto se acceleriamo il processo di convergenza è chiaro che prima raggiungeremo la condizione di arresto e questo accadrà prima del raggiungimento del numero massimo di iterazioni. In questo caso abbiamo utilizzato una tolleranza sul residuo pari a $10^{-6}$, che è il valore di default, ma nella pratica questo valore può essere aumentato a seguito di alcune considerazioni fisiche. Abbiamo avuto modo di sperimentare che in ambito Photometric Stereo è possibile aumentare tale parametro sino a $10^{-1}$ e questo principalmente per due motivi. Infatti il nostro occhio non è in grado di distinguere i dettagli oltre a $10^{-2}$, quindi visivamente parlando, non saremmo in grado di cogliere le differenze di una soluzione calcolata con tolleranza inferiore. Inoltre i nostri dati sono fortemente mal condizionati pertanto qualsiasi sia il valore di tolleranza il valore del residuo si manterrà pressoché costante. Tale scelta quindi ci consente di ridurre ulteriormente il numero di iterazioni e quindi di conseguenza i tempi di calcolo.

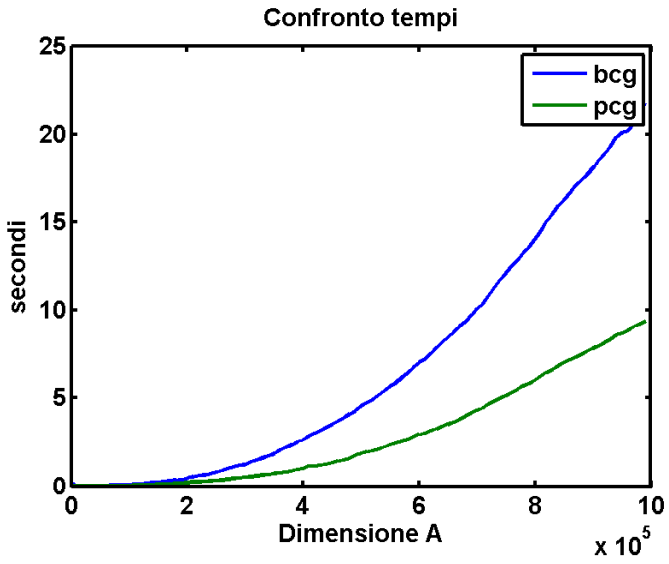
In questa seconda parte di test confronteremo il metodo del gradiente coniugato con il metodo del gradiente biconiugato (bcg). Questo perché nella storia della Photometric Stereo, quest'ultimo è sempre stato utilizzato come metodo di integrazione principale. A differenza però del pcg, il bcg risulta più lento in quanto il numero delle operazioni che vengono eseguite sono doppie rispetto al pcg. Infatti ad ogni passo vengono svolti due prodotti differentemente al pcg in cui il prodotto è uno solo. E' altresì vero che il bcg come specifica di ingresso non richiede matrici definite positive, per cui trova applicazione su un più ampio raggio di problemi. Anche in questo caso Matlab mette a disposizione una funzione `bicg`, e i parametri di ingresso e di uscita sono i medesimi al pcg.

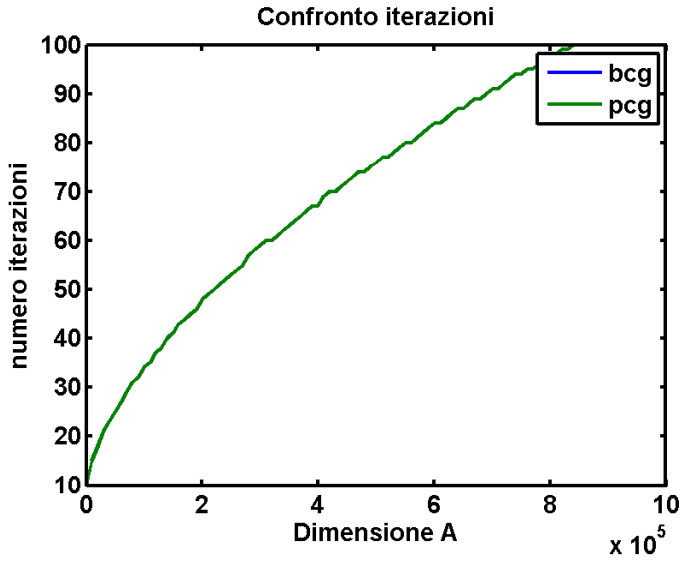
Effettueremo gli stessi test fatti in precedenza, precondizionando entrambi i metodi col medesimo precondizionatore e lasciando inalterati i parametri di tolleranza e numero massimo di iterazioni.




A livello di precisione i due metodi si comportano allo sesso identico modo. Il grafico mostra la curva verde in quanto quella blu è perfettamente sovrapposta e se si calcola la norma infinito della differenza delle due si ottiene perfettamente zero. Inoltre l'ordine di grandezza del residuo si mantiene su $10^{-6}$, il che ci suggerisce che i metodi stanno convergendo.



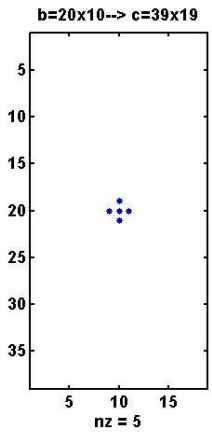



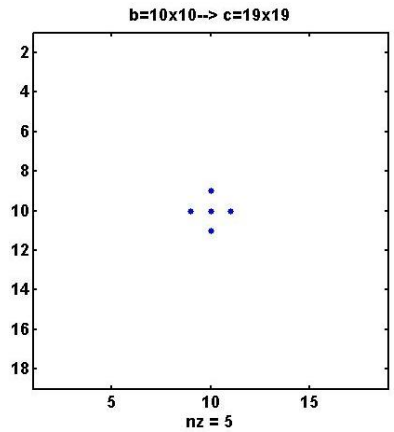

Anche il numero delle iterazioni è identico in entrambe i casi. La differenza sostanziale però la si registra poi sui tempi. Infatti con il bcg impieghiamo esattamente il doppio del tempo per risolvere il sistema, e questo in accordo a quanto detto precedentemente, ovvero che vengono eseguite il doppio delle operazioni. Pertanto nel caso in esame, dal momento che la nostra matrice è sempre definita positiva, il pcg è preferibile rispetto al bcg in quanto otteniamo gli stessi risultati ma in un tempo dimezzato rispetto al bcg.

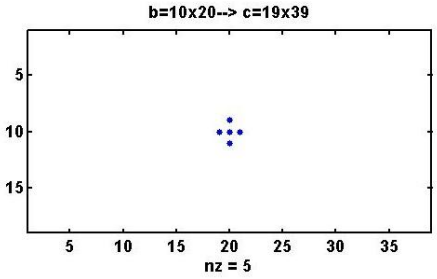
## 3.11 Prodotto Toeplitz

Abbiamo dimostrato nel paragrafo 3.5.2 come la matrice di Poisson sia una matrice di Toeplitz. E questo l'abbiamo fatto in due modi, osservando che tutti i termini nelle diagonali rimangono costanti e richiamando il concetto di matrice multi indice. Le matrici di Toeplitz sono oggi giorno uno degli argomenti di maggior interesse algebrico [3] ed è tutt'ora un campo ancora in piena evoluzione ed espansione, motivo per il quale ci siamo voluti cimentare per provare a risolvere il nostro sistema di Poisson. In generale quando si ha a che fare con matrici Toeplitz è possibile svolgere il prodotto matriciale $T * x$ in modo molto rapito utilizzando la trasformata di Fuorier veloce (fft). Per far ciò si definisce una funzione prodotto che ha la caratteristica di alterare la matrice di partenza generandone un'altra del tutto analoga che è strutturata in maniera tale da consentire un prodotto più rapido. Tale trasformazione in letteratura matematica moderna prende nome di *gnomone* (notazione:$a$) che ci accingiamo a costruire. Se la nostra immagine pertanto è di dimensione $m \times n$ la matrice gnomone risultate avrà dimensione $2m-1 \times 2n-1$. Quest'ultima è strutturata in modo tale da avere al centro una regione in cui sono contenuti tutti i termini non nulli delle diagonali della matrice di Poisson. Quindi l'elemento centrale sarà $4$, e sarà circondato dai termini unitari che costituiscono le altre diagonali della matrice di partenza. In altri termini lo gnomone sarà così strutturato:

$$\begin{pmatrix} 0 & & \cdots & & 0 \\ & 0 & -1 & 0 & \\ \vdots & -1 & 4 & -1 & \vdots \\ & 0 & -1 & 0 & \\ 0 & & \cdots & & 0 \end{pmatrix} \qquad (3.39)$$




Interessante constatare come tale matrice risulti molto compatta a differenza della matrice di Poisson. Infatti se la nostra immagine avesse una dimensione per esempio $16 \times 16$ la dimensione dello gnomone sarebbe $31 \times 31$. Mentre nel caso utilizzassimo la matrice di Poisson ovviamente l'immagine deve essere necessariamente vettorializzata e la dimensione della matrice risultante è pari a: $256 \times 256$. In termini di occupazione di spazio di memoria cambia poco, e questo perché la matrice di Poisson è sparsa mentre lo gnomone è una matrice piena in cui memorizziamo anche gli elementi nulli.

Una volta quindi costruito lo gnomone possiamo procedere risolvendo il prodotto come:

$$b = \mathcal{F}^{-1}\big(\mathcal{F}(a * x)\big) = \mathcal{F}^{-1}\big(\mathcal{F}(a) * \mathcal{F}(x)\big) \qquad (3.40)$$

In Matlab la (3.40) può essere implementata nel seguente modo: `b = ifft(fft(a).*fft(x))`.
Ma tale funzione ci consente solo di svolgere il prodotto e non di risolvere il sistema che deve essere ancora risolto. Per far ciò ancora una volta possiamo ricorrere al pcg in quanto $a$ anche in questo caso è definita positiva. In realtà per le sperimentazioni si è fatto uso di un software completo realizzato dal Prof. G. Rodriguez [4] (tpgc) che allo stesso tempo integra la parte del prodotto toeplitz con la risoluzione del sistema mediante pcg. Anche in questo caso ovviamente abbiamo fatto uso di un precondizionatore, però differente rispetto a quello visto in precedenza. Infatti questa volta la matrice di ingresso è lo gnomone, che si è sparso ma presenta una struttura totalmente differente rispetto a Poisson. Per cui abbiamo optato per il precondizionatore di Chan, una matrice circolante e per tale caratteristica come parametro di ingresso è sufficiente passare solo la prima colonna, in quanto per le proprietà delle matrici circolanti una volta nota la prima colonna è possibile ricostruire l'intera matrice [3][1]. In questo caso preferiamo non mostrarvi risultati, che rimandiamo al test finale in cui confronteremo tutti metodi analizzati.

## 3.12 Costruzione del software di integrazione

Come abbiamo fatto per il caso dei vari algoritmi sintetizzati sui gradienti anche in questo caso, per completezza, mostreremo il codice sorgente Matlab del software che abbiamo realizzato per l'integrazione.

```
function[zz k res]=integration(p,q,mask,met,tol,maxit)


if nargin<6,  maxit=100, end
if nargin<5,  tol=10^-1, end
if nargin<4,  met=1, end  %fastest%

[m,n]=size(p),
[m1,n1]=size(q),
[m2 n2]=size(mask),

if(m2~=m || n2~=n || m2~=m1 || n2~=n1)
error('mask and gradient must have the same dimension'),
end

p=p./max(max(p)),
q=q./max(max(q)),


ro=zeros(m,n),

%disp('building ro')

for i=1:m-1
    for j=1:n-1
   if mask(i,j)==1
```

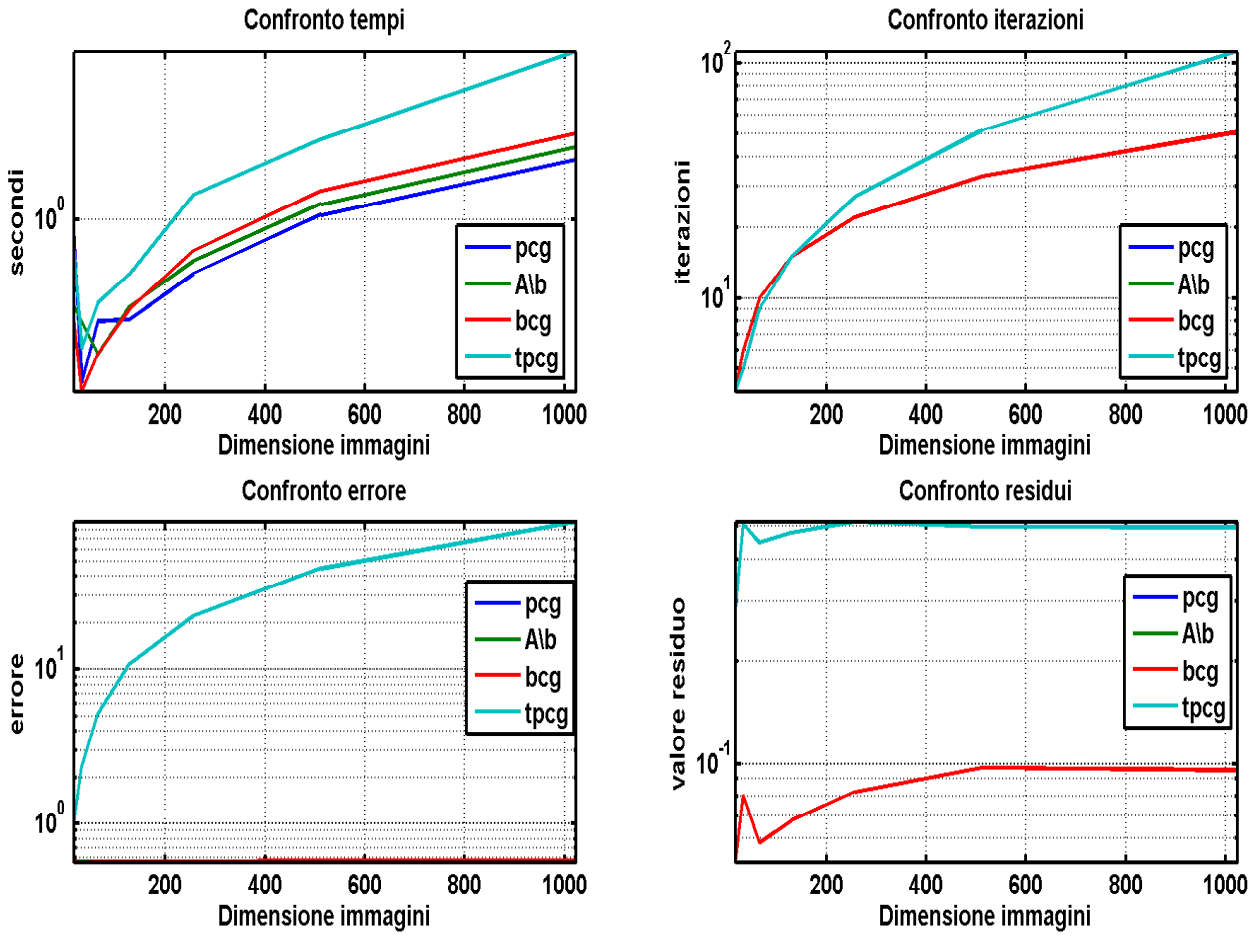
```
        ro(i,j)=p(i+1,j)-p(i,j)+q(i,j+1)-q(i,j)+eps,
    else

      ro(i,j)=0,

   end

    end
end

%The matrix A is square (m=n).Contrarily ro could be rectangular(m!=n).
%So we need to convert in a square matrix adding zeros.
if (met==1 || met==2 || met==4)
maxx=max(m,n),
m3=maxx-m,
n3=maxx-n,
ro(m+m3,n+n3)=0,
A=gallery('poisson',maxx),
ro(isnan(ro))=0,
ro(isinf(ro))=0,

ro=reshape(ro,maxx.^2,1),
end

switch met
case 1
%disp('pcg(A,ro)')

L=ichol(A,struct('michol','on')),
[z,~,res, k]=pcg(A,(-ro),tol,maxit,L,L'),


case 2
%disp('A\ro')
z=-A\ro,
k=0,
res=0,
case 3
%disp('tpcg')
[m3 n3]=size(ro),
a=zeros(2*m3-1,2*n3-1),
a(m3,n3)=4,
a(m3,n3+1)=-1,
a(m3,n3-1)=-1,
a(m3+1,n3)=-1,
a(m3-1,n3)=-1,
c=tpchan2(a),
x0=zeros(m3,n3),
[z k res]=tpcg2(a,-ro,c,0,x0,tol,maxit),
res=res(end),

case 4
%disp('bicg(A,ro)')
L=ichol(A,struct('michol','on')),
[z,~,res,k]=bicg(A,(-ro),tol,maxit,L,L'),
otherwise
error('uknown method')

end
```

```
%zz now is the solution but it's a vector! so we need to reshape in to a
%matrix

if (met==1 || met==2 || met==4)
zz=reshape(z,maxx,maxx),
zz=zz(1:m,1:n),
zz=zz/max(max(zz)),

else
    zz=reshape(z,m3,n3),

end
%cleaning part!! in this section we're going to apply the mask to the solution
to delight any noise part around object surf

 for i=1:m
    for j=1:n

       if(mask(i,j)==0)

           zz(i,j)=0,
       end
    end
end

end
```

La funzione integration prende in ingresso i gradienti, la maschera e un parametro *met* che consente di poter scegliere il metodo di integrazione. Inoltre è possibile inserire il numero massimo delle iterazioni e la tolleranza. La funzione in output restituisce la superficie *zz*, il numero di iterazioni e il residuo, informazioni che abbiamo utilizzato per mostrare i risultati sui vari test. Dopo una fase iniziale in cui si verifica la consistenza dei dati in ingresso, costruiamo il termine noto che sarà una matrice di dimensione pari ai gradienti. Se scegliamo come metodo di integrazione Gauss, pcg o bcg, tale vettore deve essere necessariamente vettorializzato, non solo la matrice di Poisson è una matrice che può essere solo quadrata, mentre il termine noto può essere anche una matrice rettangolare. Pertanto prima di effettuare l'operazione di reshape è necessario rendere quadrato il termine noto e questo viene fatto semplicemente aggiungendo degli zeri. Se invece la modalità di integrazione scelta è il tpcg non è necessario ne quadrare ne vettorializzare il termine noto. Quindi i vari metodi di integrazione vengono gestiti mediante una struttura switch che selezionerà la modalità in base al valore di *met* inserito in ingresso. Se integriamo con Gauss, pcg o bcg la soluzione risultante sarà quindi un vettore. Per avere dunque la superficie sarà sufficiente effettuare un reshape finale. Al contrario la soluzione calcolata con tpcg coincide con la superficie finale per cui non è necessario effettuare alcuna ulteriore operazione. Tutte questi processi vengono controllati da una condizione if che eseguirà le operazioni appena indicate a seconda del metodo di integrazione scelto. Infine si effettua un'operazione di pulitura riapplicando nuovamente la maschera alla soluzione finale, che quindi sarà finalmente pronta per essere visualizzata.

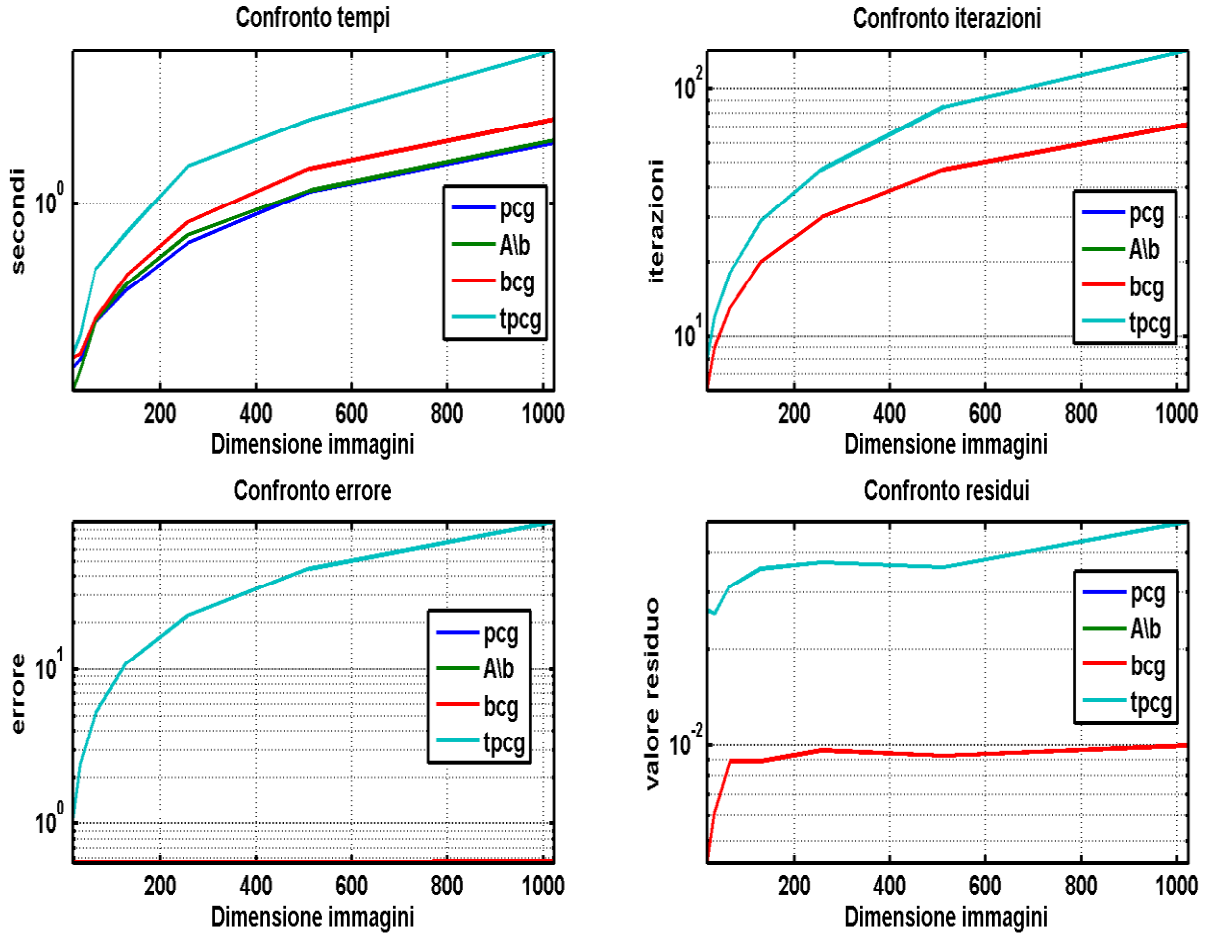
## 3.13 Confronto dei vari metodi di integrazione

In questo paragrafo conclusivo metteremo a confronto tutti i vari metodi di integrazione descritti in questo capitolo. Per far ciò abbiamo costruito un banco di prova riferendoci al modello sintetizzato descritto in 2.6.2. Quindi come fatto nei test precedenti, misureremo i tempi di calcolo, i valori di residuo, il numero delle iterazioni e l'errore stimato sulla superficie vera, che conosciamo a priori dal momento che stiamo facendo uso del modello sintetizzato. Quest'ultimi verranno ricavati iterativamente al crescere della dimensione dei gradienti (e quindi delle immagini che compongono il data set). A differenza dei test effettuati precedentemente, in cui ci siamo limitati a risolvere i sistemi lineari semplicemente richiamando le varie routine matlab, questa volta faremo uso del software da noi realizzato e descritto nel paragrafo precedente, in cui ovviamente i tempi di calcolo non tengono conto solo della risoluzione del sistema, ma anche ovviamente della costruzione del termine noto e di tutte le altre varie componenti. Inizieremo dapprima utilizzando una tolleranza per i metodi iterativi di $10^{-1}$, e poi ripeteremo lo stesso test con un valore pari a $10^{-2}$.




In termini di rapidità il pcg si rileva il più rapido di tutti seguito in ordine dal metodo di Gauss, bcg e tpcg, che in generale si mantiene più lento di circa un ordine di grandezza rispetto a tutti gli altri. Questo fatto lo si evince anche dal numero di iterazioni che è decisamente superiore rispetto a bcg e pcg, che anche in questo caso presentano lo stesso numero di iterazioni (mentre nel metodo di Gauss il numero di iterazioni è sempre evidentemente nullo dal momento che abbiamo a che fare

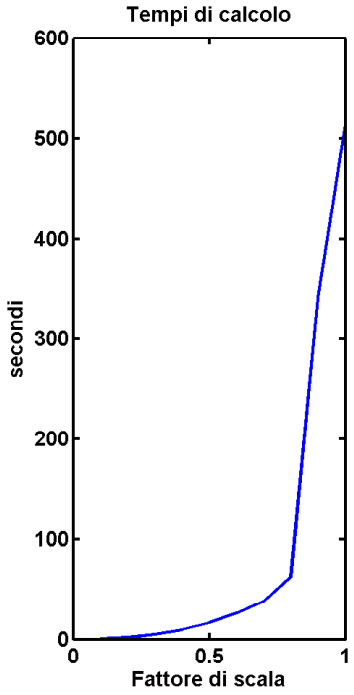
con un metodo diretto). La spiegazione di questo fatto è e abbastanza evidente, in quanto la proprietà di matrice a banda è predominante rispetto alla proprietà Toeplitz. Anche in termine di precisione il tpcg risulta più scadente, a differenza di tutti gli altri metodi, che si comportano sostanzialmente allo stesso modo. Infatti l'errore stimato sulla soluzione vera tende crescere in modo lineare all'aumentare della dimensione delle immagini (su scala lineare, i grafici riportati sono in scala semilogaritimica per una migliore visualizzazione). Questo trend risulta meno evidente se si valutano i residui che in tutti i casi assumono un andamento pressoché costante, rimangono comunque più elevati quelli riscontrati utilizzando il tpcg. A questo punto possiamo passare all'analisi dei risultati ottenuti con tolleranza inferiore.

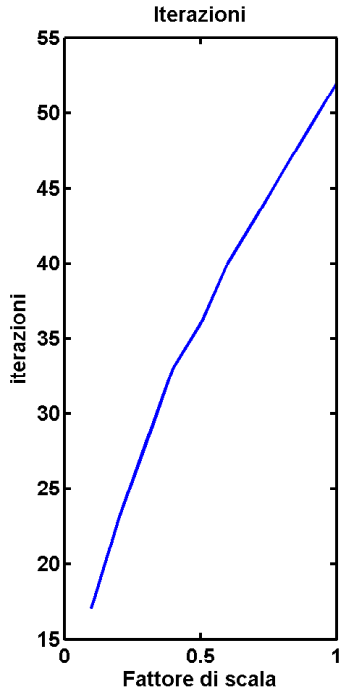


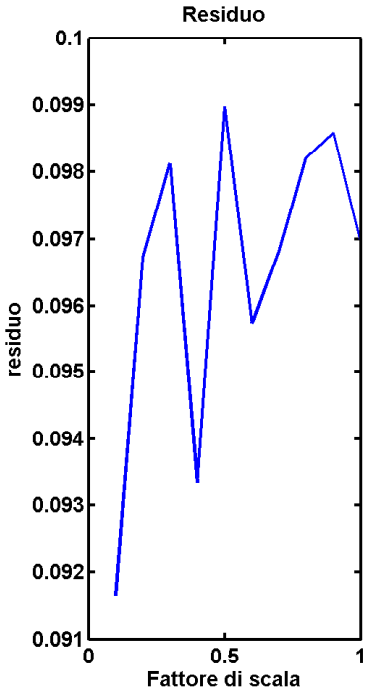

E' immediato constatare come il valore del residuo sia in questo caso un ordine di grandezza inferiore rispetto a prima, il che è del tutto lecito dal momento che in questo caso abbiamo utilizzato una tolleranza di $10^{-2}$. Ma in termini di errore i risultati ottenuti sono i medesimi, infatti se si calcola la differenza con l'errore stimato precedentemente quest'ultima è identicamente nulla. Il fatto però di utilizzare un valore di tolleranza inferiore implica un aumento generale del numero di iterazioni e quindi di tempo di calcolo, e ciò sarà vero tanto più grandi saranno le dimensioni delle immagini con cui stiamo lavorando. In altri termini nella pratica scegliere un valore di tolleranza più alto non è poi così lesivo nella qualità delle ricostruzioni che otteniamo, e allo stesso tempo tale scelta ci consente di abbattere drasticamente il numero delle iterazioni. Per completezza abbiamo svolto gli stessi test su un architettura fissa intel-core7 con S.O. linux ottenendo praticamente gli stessi risultati, a conferma del fatto che l'architettura incide poco sulle prestazioni.

### 3.13.2 Test pcg effettuato su data set vero

Per concludere definitivamente il discorso sui vari metodi di integrazione vi proponiamo un test finale questa volta realizzato su dati veri. Il data set è composto da quattro immagini in formato tif, in cui ovviamente come primo passaggio estrarremo i gradienti utilizzando i metodi descritti nel capitolo due. Successivamente daremo quindi in pasto i gradienti elaborati al nostro software di integrazione limitandoci però questa volta ad utilizzare il solo metodo pcg, che abbiamo avuto modo di constatare essere il migliore in termini di rapporto precisione velocità. Ovviamente le immagini sono state acquisite in piena risoluzione, pertanto ripeteremo il processo di integrazione più volte al crescere delle dimensioni delle immagini partendo da un fattore di scala pari a 0.1 sino ad arrivare a 1 (che coincide con la dimensione naturale delle immagini). Anche questa volta misureremo i tempi di calcolo, il numero delle iterazioni e il valore del residuo. Non potremmo invece stimare l'errore sulla soluzione vera dal momento che questa volta non conosciamo la superficie reale.

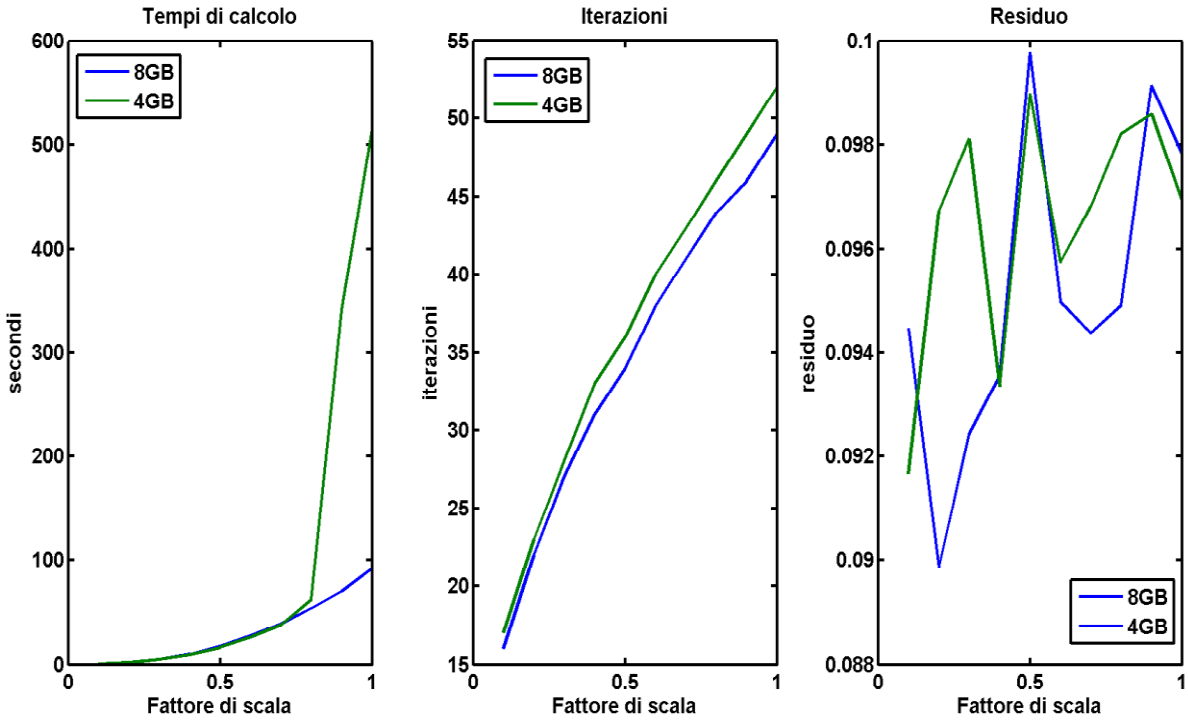



| Fattore di scala | Dimensione immagine |
|---|---|
| **0.1** | 285x429 |
| **0.2** | 570x858 |
| **0.3** | 855x1287 |
| **0.4** | 1140x1716 |
| **0.5** | 1424x2144 |
| **0.6** | 1709x2573 |
| **0.7** | 1994x3002 |
| **0.8** | 2279x3431 |
| **0.9** | 2564x3860 |
| **1** | 2848x4288 |

L'andamento dei tempi si mantiene sostanzialmente lineare sino a quando raggiungiamo il fattore di scala 0.8 in cui si registra un brusco innalzamento. Questo in realtà è dovuto ai limiti di architettura macchina. Infatti in questo caso disponevamo di soli 4GB di RAM indirizzabili, che evidentemente non sono più sufficienti per fattori di scala superiori 0.8, motivo per il quale parte dell'indirizzamento avviene su disco fisso, che è spaventosamente più lento in termini di accesso e scrittura. Nonostante ciò i risultati sono comunque sorprendenti in quanto riusciamo a lavorare a piena risoluzione superando di poco i 500 secondi (circa 8 minuti). Ottimi risultati si riscontrano anche in termini di numero di iterazioni, infatti il valore massimo non supera le 60. Il che significa che il precondizionatore si sta comportando piuttosto bene. Anche i risultati in termini di residuo sono coerenti con la scelta di aver utilizzato anche in questo caso una tolleranza pari a $10^{-1}$.

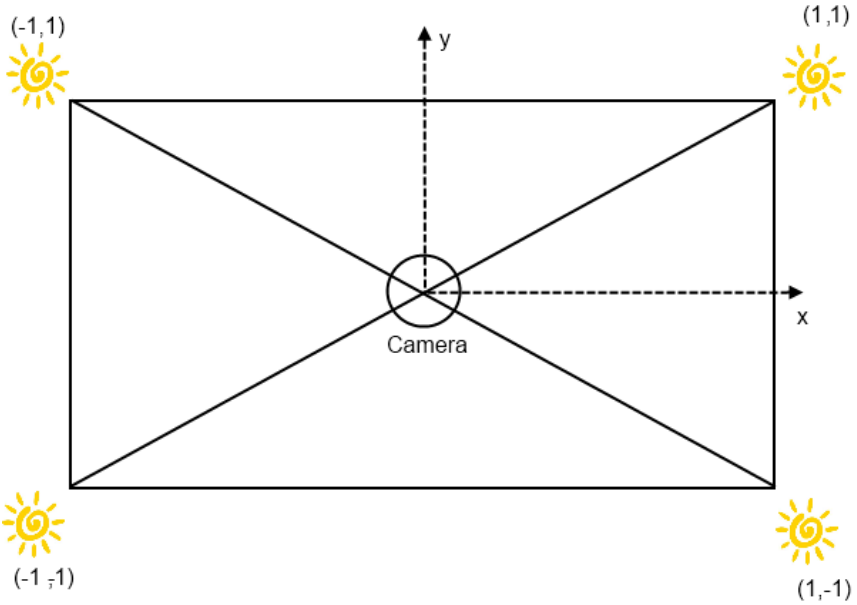



A conferma di quanto dicevamo in precedenza, se ripetiamo il medesimo test su una architettura fissa con processore i-7 con 8GB di RAM indirizzabili, non si registra il brusco innalzamento dovuto all'accesso sul disco. Il che ci suggerisce che per una rapida analisi sul campo, se non si predispone di calcolatori particolarmente performanti, conviene lavorare a risoluzione medie che superano comunque i mille pixel, (pertanto comunque risoluzioni di tutto rispetto), e rimandare l'elaborazione a piena risoluzione in un secondo momento utilizzando calcolatori dotati di una quantità sufficiente di RAM indirizzabile.

# 4 Risultati numerici

## 4.1 Introduzione

In questo breve capitolo mostreremo graficamente alcuni risultati ottenuti con i nostri software di elaborazione. Utilizzeremo due serie di data set differenti. Il primo è caratterizzato da immagini puramente archeologiche acquisite in parte in Val Camonica e ad Anela (Sardegna) dalla Dott.ssa Carla Mannu [13]. I data set in questione raffigurano per lo più incisioni su muri e rocce e sono composti da quattro immagini ciascuno, acquisite applicando un flash in quattro direzioni differenti. Di seguito riportiamo il sistema di acquisizione utilizzato.




A partire dalla conoscenza della posizione delle sorgenti possiamo ricavare la matrice delle direzioni delle luci, che in tutti i casi si presenterà nella forma:

$$L = \begin{bmatrix} 1 & -1 & -\rho \\ 1 & 1 & -\rho \\ -1 & 1 & -\rho \\ -1 & -1 & -\rho \end{bmatrix}$$

In cui $\rho$ rappresenta la coordinata lungo l'asse z è può essere scelta arbitrariamente. In realtà per metterci nelle condizioni di idealità quest'ultima dovrebbe valere infinito o quanto meno un valore elevato che approssimi la posizione della luce tale da poter essere considerata puntiforme. Dal momento che conosciamo le direzioni luce, per questa serie di data set saremo in grado di effettuare due ricostruzioni utilizzando i due diversi metodi di elaborazione dei gradienti analizzati nel capitolo 2 (metodo di elaborazione standard, 4-harmonics).
In una seconda fase invece mostreremo alcuni risultati utilizzando dei data set caratterizzati da dodici immagini ciascuno che abbiamo reperito in rete. In questo caso purtroppo non disponiamo della matrice di direzione luce in quanto ovviamente non conosciamo il processo di acquisizione che è stato adottato. Pertanto ci limiteremo ad effettuare delle ricostruzioni basate esclusivamente sul metodo di elaborazione dei gradienti 4-harmonics.

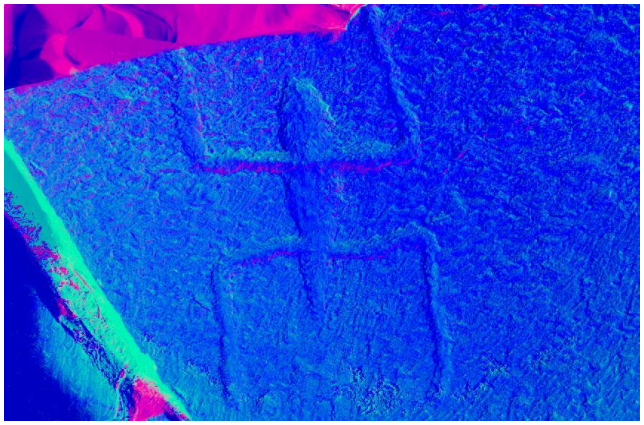
### 4.1.2 Risultati ottenuti su data set archeologici

Di seguito riportiamo i risultati grafici ottenuti utilizzando i data set archeologici acquisiti utilizzando il sistema descritto precedentemente. Per ciascuno mostreremo per compattezza una sola delle immagini originali di acquisizione, l'albedo, la mappa delle normali e ovviamente le ricostruzioni ottenute per integrazione a partire dalla conoscenza dei gradienti $p$ e $q$. Il tutto verrà ripetuto due volte utilizzando in un caso il metodo di elaborazione dei gradienti standard e successivamente il 4-harminics.

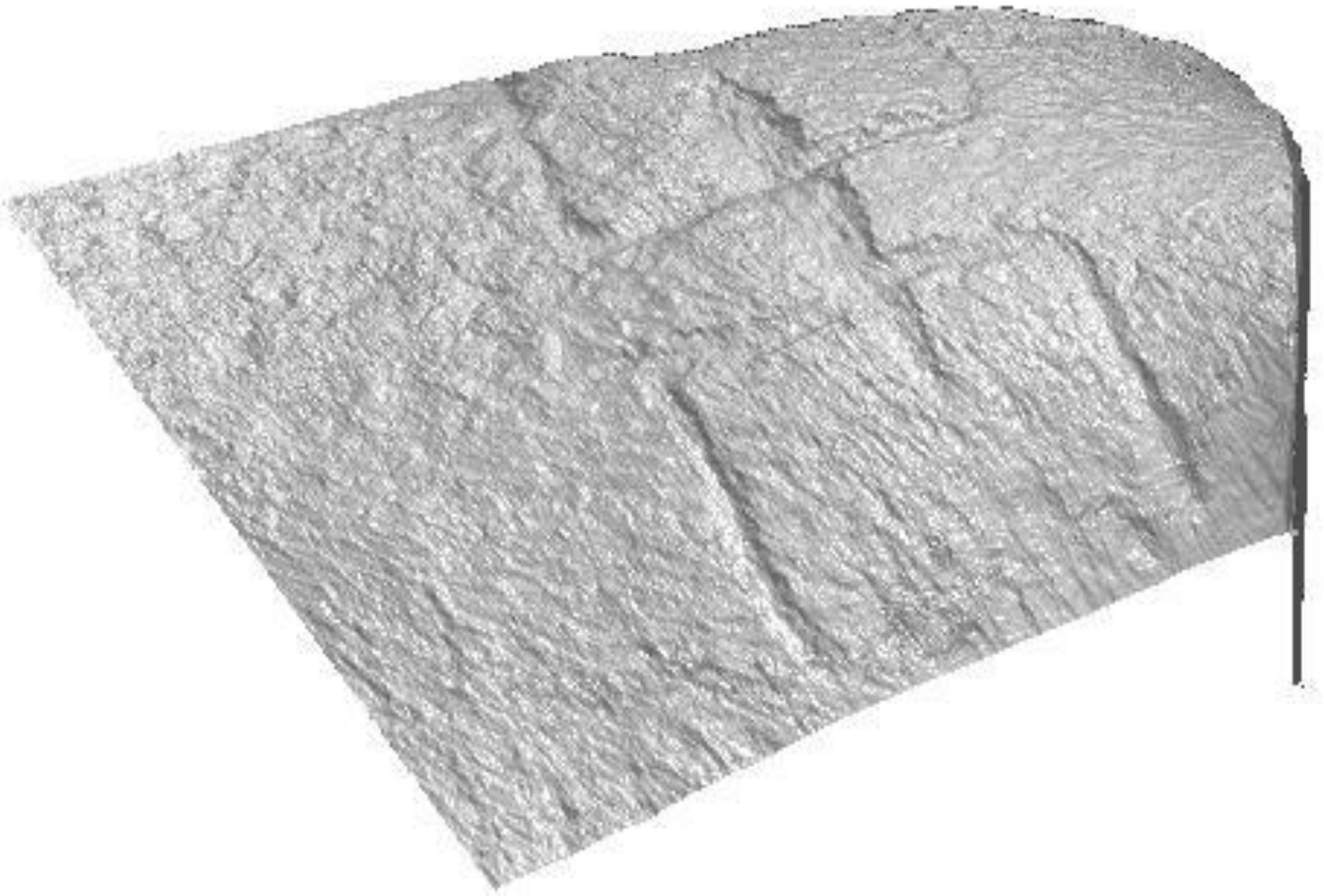
- **Data set 1**

Risultati ottenuti con il metodo di elaborazione dei gradienti standard.

*Albedo*

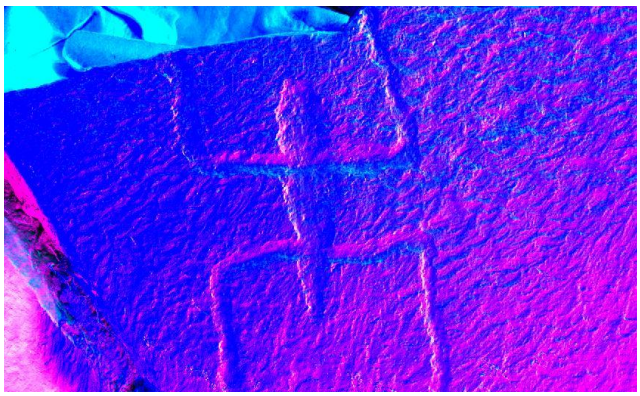


*Normale*

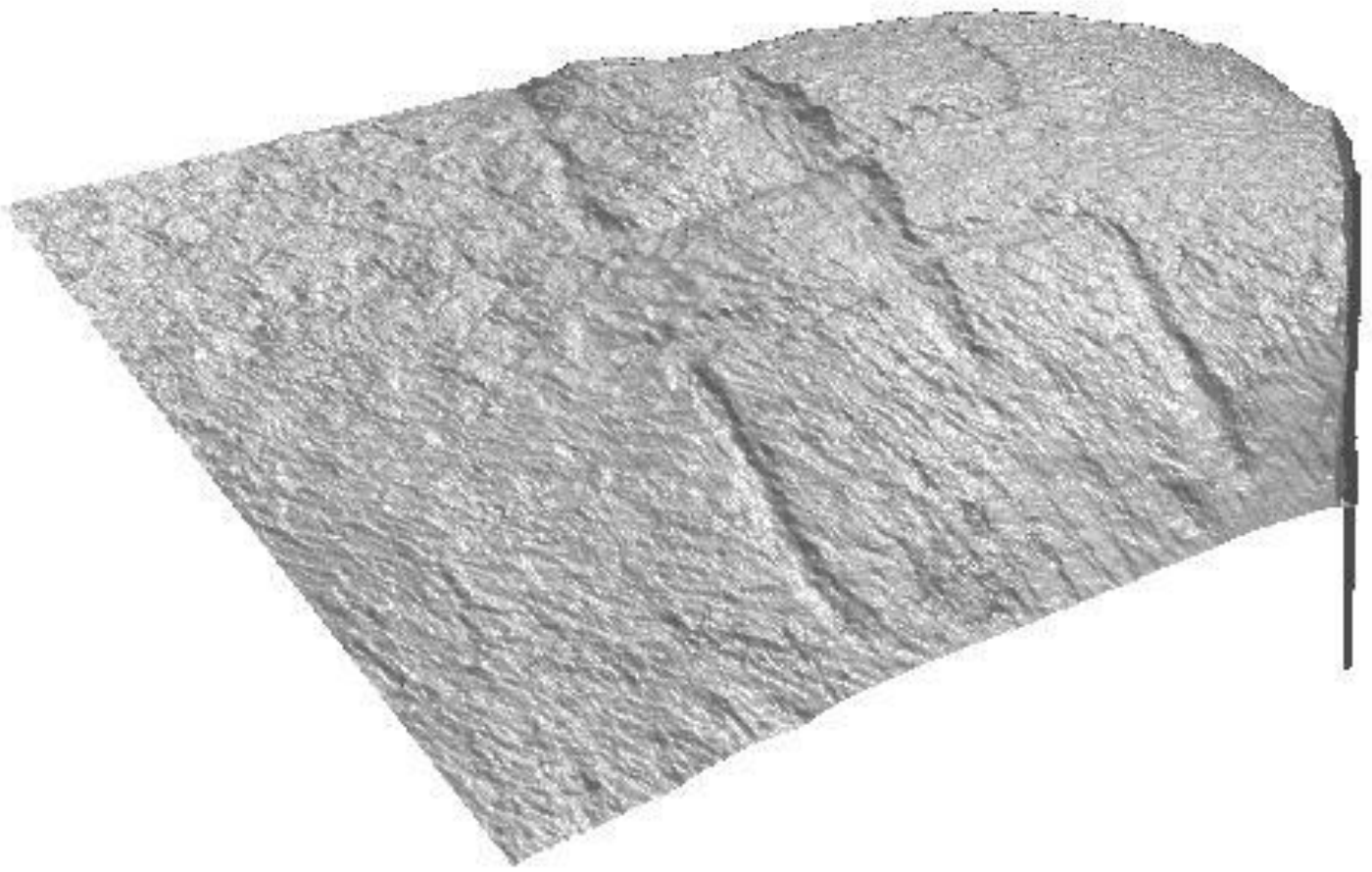


Risultati ottenuti con il metodo di elaborazione dei gradienti 4-harmonics.

*Albedo*



*Normale*



- **Data set 2**

Risultati ottenuti con il metodo di elaborazione dei gradienti standard.

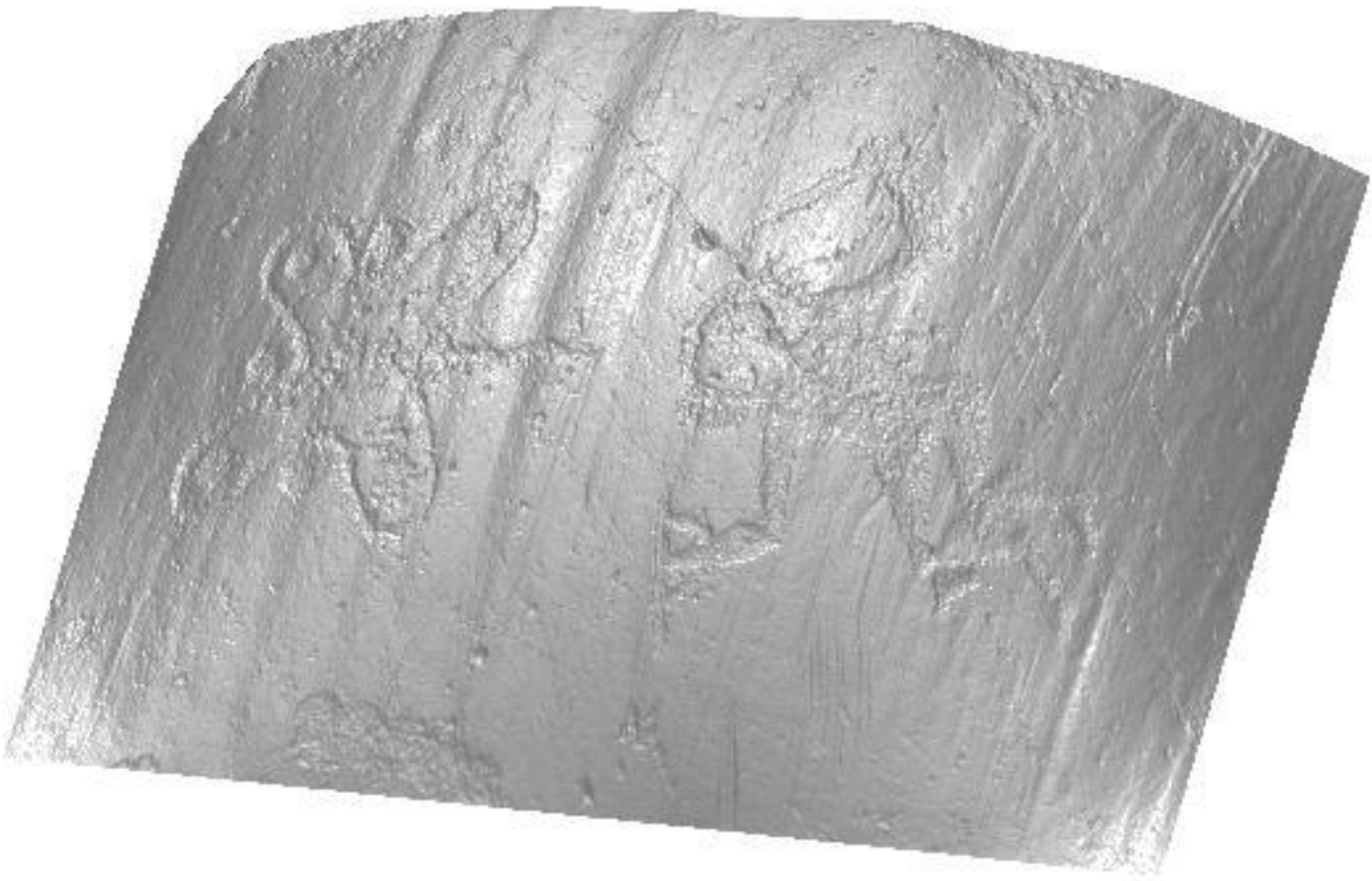
*Albedo*

*Normale*

Risultati ottenuti con il metodo di elaborazione dei gradienti 4-harmonics.

*Albedo*

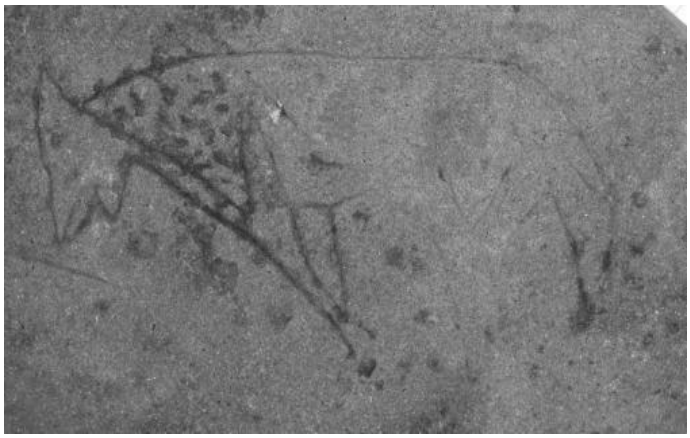
*Normale*



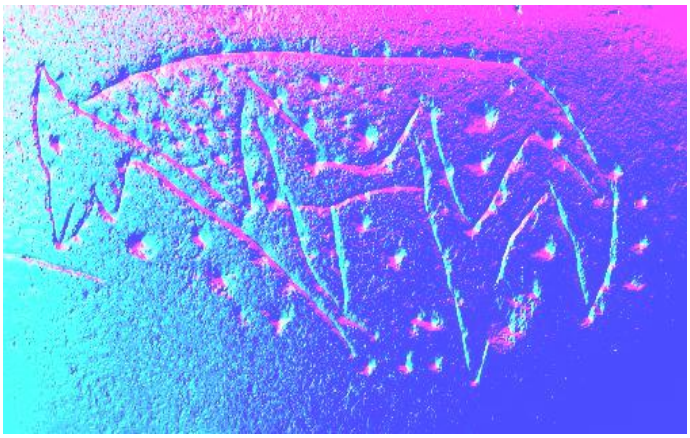
- **Data set 3**

Di proprietà del Prof. Paolo Emilio Bagnoli che ringraziamo per la concessione(Grotta Polesini, Tivoli)

Risultati ottenuti con il metodo di elaborazione dei gradienti standard.



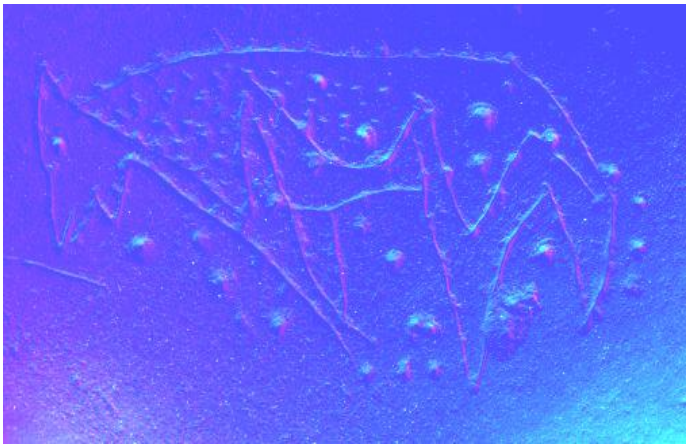
*Albedo*



*Normale*

Risultati ottenuti con il metodo di elaborazione dei gradienti 4-harmonics.

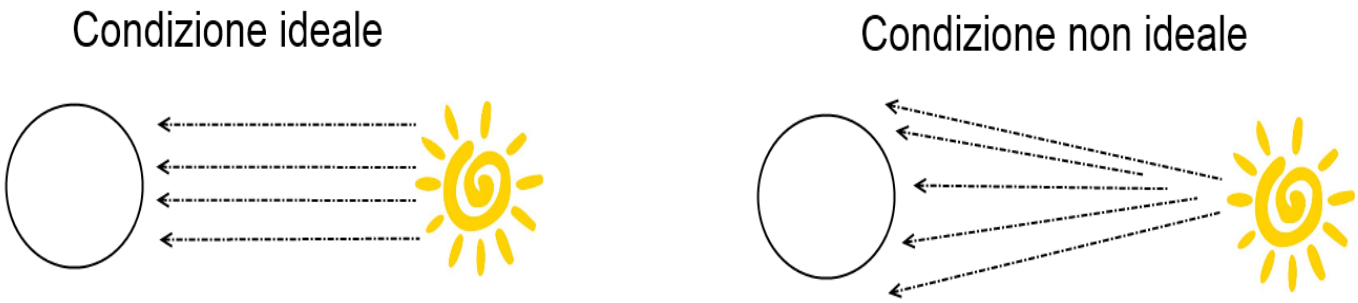
*Albedo*



*Normale*

### 4.1.3 Convex effect

Come avrete notato le ricostruzioni sono piuttosto fedeli rispetto alle immagini di partenza, il che non è del tutto scontato dal momento che abbiamo a che fare con data set in cui intervengono tutti i fattori di non idealità possibili. Sicuramente il fattore che interviene però con maggiore criticità è la distorsione legata al fenomeno di non idealità delle nostre sorgenti. Infatti se consideriamo la nostra superficie come il nostro sistema ricevente, quest'ultimo non vede le sorgenti come puntiformi. Infatti quest'ultime non sono posizionate all'infinito ma a una distanza di qualche metro/centimetro dall'oggetto. Il che implica che i raggi che incidono sulla superficie non sono perfettamente paralleli. Questo fenomeno purtroppo genera un effetto bombato nelle ricostruzioni che ha destato delle perplessità in fase di analisi da parte dei nostri colleghi archeologi.

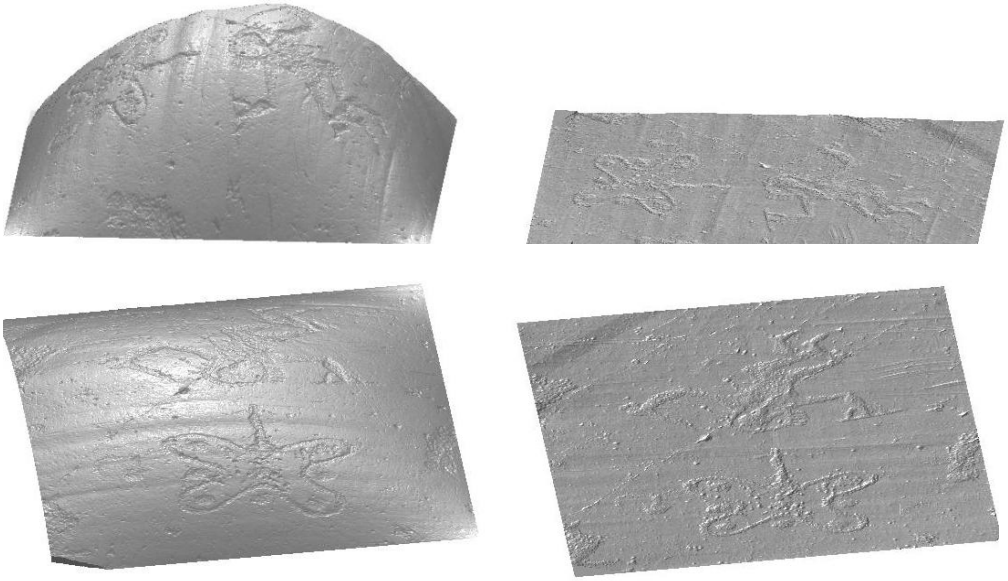



Tale effetto, a nostro modo di vedere, potrebbe essere controllato utilizzando due precauzioni. La prima è di carattere puramente pratico, infatti si potrebbe pensare di distanziare il sistema di acquisizione a una distanza tale da poter considerare la sorgente come puntiforme, in modo tale che i raggi luminosi risultino il più paralleli possibile. Ovviamente questa operazione non è sempre possibile e ciò è in funzione essenzialmente del luogo e delle dimensioni della superficie da ricostruire. Infatti se quest'ultima fosse molto grande rispetto al sistema di acquisizione implicherebbe posizionare le sorgenti a una distanza tale per cui sarebbe necessario l'utilizzo di zoom ottico che nuovamente ha l'effetto poi di introdurre altre distorsioni, non solo ma nella maggior parte dei casi le aree di acquisizione sono piccole e strette pertanto la posizione del sistema è poco ingegnerizzabile. L'altra precauzione può essere invece presa a livello implementativo infatti la nostra superficie perturbata può essere vista come: $z = f(x, y) + \omega(x, y)$. Dove $\omega(x, y)$ rappresenta la nostra funzione di distorsione, che una volta nota, può essere utilizzata per effettuare un processo di regolarizzazione geometrica. Il problema è che per ottenere buoni risultati questa funzione deve essere nota con una certa precisione il che non è un'operazione affatto semplice. Per ricavarla si potrebbe pensare di lavorare nel dominio della frequenza e quindi utilizzare l'analisi di Fourier. Questo processo è evidentemente una fase di ottimizzazione piuttosto complessa che sicuramente merita tutta la nostra attenzione ma in parte esula dagli obbiettivi di questo testo. Nonostante ciò vi proponiamo una soluzione, piuttosto semplice, che abbiamo ideato per far fronte al problema, da cui abbiamo ottenuto risultati interessanti per il quale vale la pena riportare. Una volta ottenuta la $z$ infatti, si può pensare di

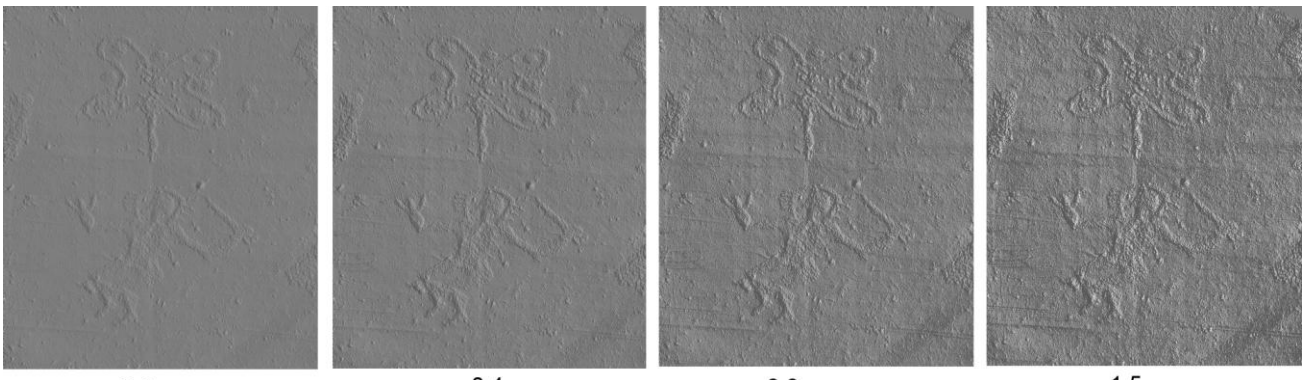
fattorizzarla mediante una opportuna decomposizione TSVD. Dal momento che siamo interessati a recuperare la sola funzione di distorsione $\omega(x, y)$, possiamo pensare di fare uso di una decomposizione troncata al solo primo valore singolare da cui ricaviamo $w$ come: $w = S_1 V_1 {D_1}^T$. Così facendo perdiamo tutti i dettagli legati alle incisioni, ma quello che rimane è proprio la forma della distorsione, ovvero ciò che ci siamo prefissati di determinare sin dall'inizio. Quindi la superficie ottimizzata sarà stimata semplicemente in termini di errore relativo come:

$$z_{ottimizzata} = \frac{|z - w|}{z}$$

L'equazione precedente ci consente di fatto di stirare le superfici elaborate ed è un'operazione che facciamo praticamente a costo zero. Infatti è un processo di ottimizzazione che viene applicato a monte, e computazionalmente parlando, anche se le $z$ di partenza assumessero dimensioni importanti, una SVD troncata al primo valore singolare è un operazione che viene svolta in tempi praticamente nulli. Di seguito riportiamo alcuni risultati grafici:



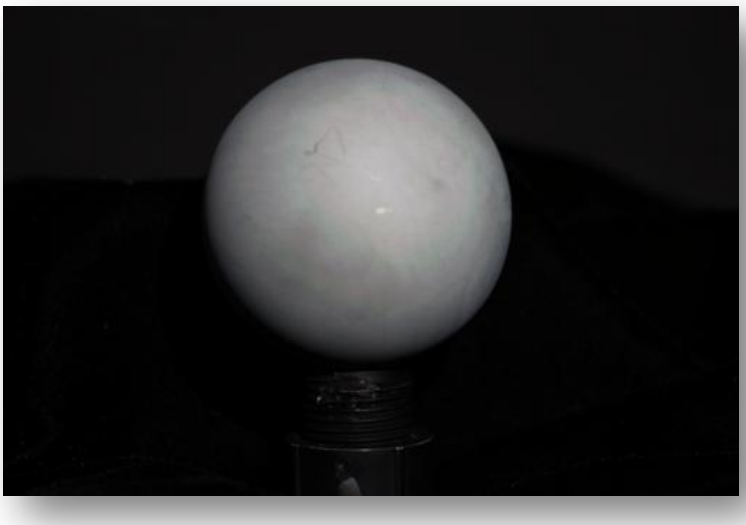
In realtà l'equazione precedente può essere modificata aggiungendo un fattore moltiplicativo di guadagno che ha l'effetto di aumentare la profondità delle incisioni e quindi di meglio evidenziare i dettagli del reperto stesso. Tale modifica porta ai seguenti risultati:

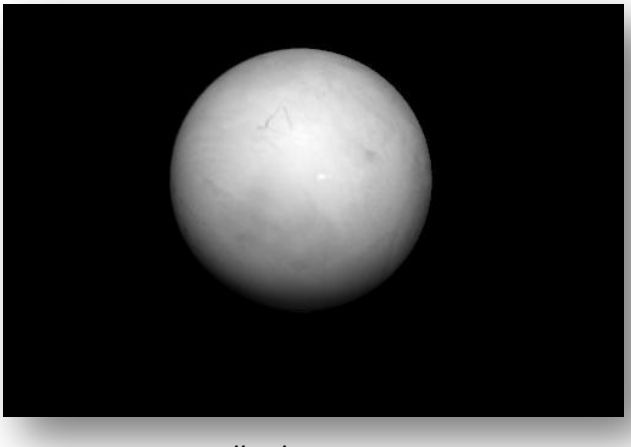


### 4.1.4 Risultati ottenuti su data set generici

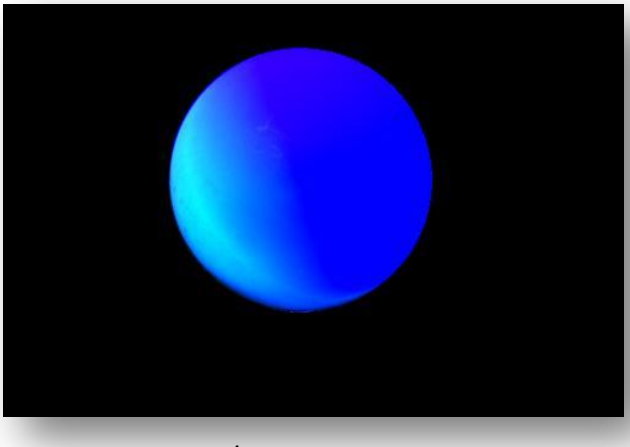
In questo sotto paragrafo riporteremo alcuni risultati ottenuti utilizzando un'altra tranche di data set che fanno riferimento come vedrete a oggetti generici. Questo in realtà per dimostrare che la Photometric Stereo, trova spazio non solo in ambito archeologico ma anche su altri settori. A differenza di quanto visto in precedenza, questa volta non disponiamo delle direzioni luce. Pertanto per l'elaborazione dei gradienti utilizzeremo il solo metodo 4-harmonics. Per la visualizzazione dei risultati manterremo la stessa profilassi adottata precedentemente, mostrando in ordine una delle immagini che compongono i vari data set e esprimendo i risultati in termini di albedo, normale e quindi ricostruzione 3-D.

- **Data set palla**



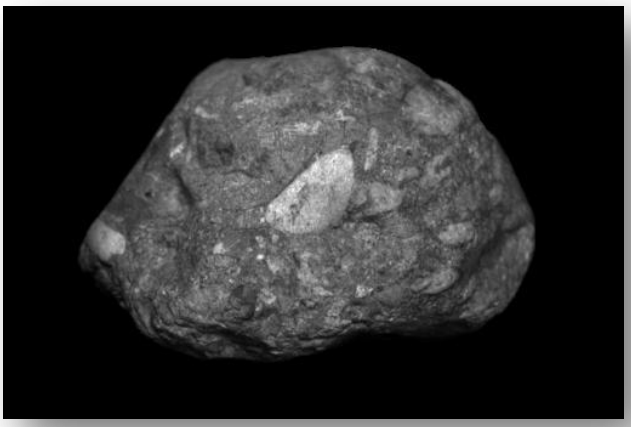


*Albedo*

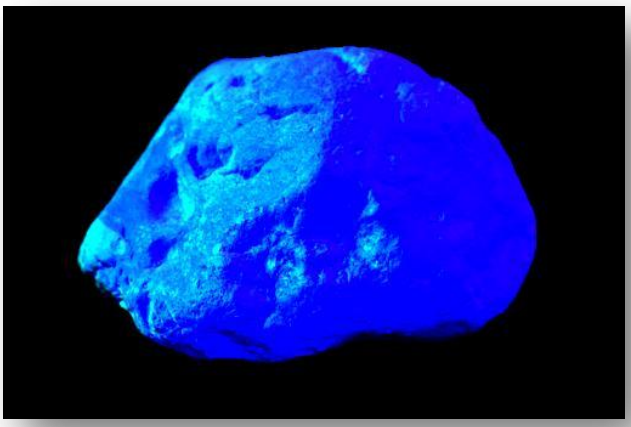


*Normale*

- **Data set roccia**



*Albedo*



*Normale*

- **Data set gatto**

*Albedo*

*Normale*

- **Data set Buddha**

*Albedo*

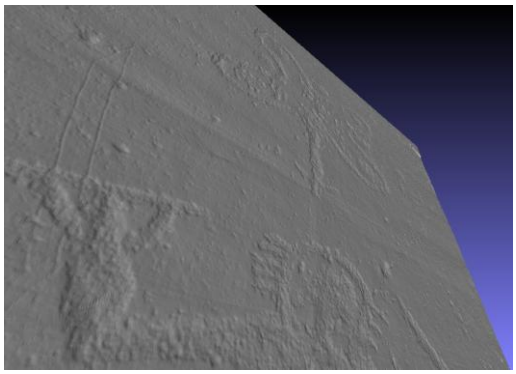
*Normale*

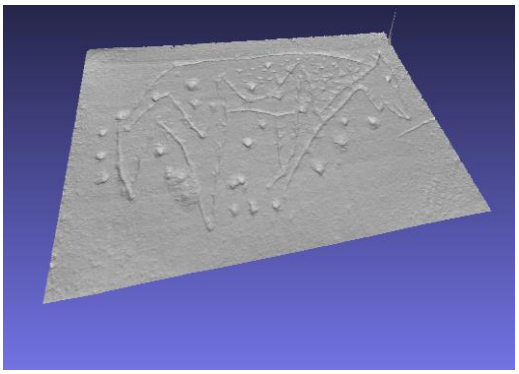
Le ricostruzioni che si ottengono anche in questo caso sono piuttosto fedeli. Sicuramente di maggiore qualità rispetto a quelle che si ottengono nel caso dei data set archeologici. E questo sicuramente in parte è dovuto al fatto di aver utilizzato un numero di immagini decisamente superiore (dodici per l'esattezza), ma soprattutto perché intervengono meno fattori di idealità. Infatti gli oggetti in questione sono molto riflettenti e approssimano molto bene la legge del coseno di Lambert. Un altro fattore di idealità importante che interviene a favore della ricostruzione, è sicuramente rappresentato dal fatto di aver immerso gli oggetti all'interno di un campo neutro (sfondo nero), che riduce enormemente la presenza di rumore sui dati in ingresso.

### 4.1.5 Software di visualizzazione delle superfici

Per tutte le illustrazioni precedenti si è fatto uso del viewer di Matlab, che è uno strumento grafico piuttosto completo equipaggiato con una serie di funzionalità: tra le principali quella di poter ruotare gli oggetti e di posizionare delle luci generiche attorno alle superfici. Ovviamente tale strumento per poter essere utilizzato implica avere installato sul proprio pc Matlab e avere chiare qualche nozione di programmazione di base, che per ovvie ragioni non fanno parte del bagaglio culturale di tutti. A tal proposito però esistono una serie di visualizzatori grafici, tra i principali (MeshLab Visual3D), costruiti con interfacce grafiche che integrano una miriade di funzionalità di visualizzazione e soprattutto sfruttano le OPEN-GLC delle schede video, consentendo una visualizzazione fluida. Tali visualizzatori supportano dei formati piuttosto standard (.3ds, .ply, .obj) che in generale sono strutturati in due sezioni. In una vengono dichiarati tutti i vertici che compongono la superficie, espressi in termini di coordinate x-y-z, in un altra vengono dichiarate le facce in riferimento ai vertici, in atri termini vengono generati dei piani che vanno a formare una struttura a poligono. Ovviamente le superfici che elaboriamo con Matlab non rispecchiano questa struttura pertanto è necessario scrivere delle opportune function che convertano il file .mat in un opportuno formato di visualizzazione 3-D. Ma ancora una volta questo può essere fatto in modo molto efficiente e rapido facendo uso della routine: surf2patch. Infatti passando in ingresso la superficie elaborata, quest'ultima restituisce due matrici che corrispondono ai vertici e alle facce. Una volta quindi ricavati questi due elementi sarà sufficiente scriverli su un file con lo standard e l'estensione caratteristica del formato di

visualizzazione ed il gioco è fatto. Di seguito riportiamo alcuni screenshot acquisiti dal visualizzatore grafico MeshLab.





## 4.2 Lavori futuri e conclusione

Con il paragrafo precedente speriamo di avervi dimostrato che è effettivamente la Photometric Stereo è un metodo di ricostruzione delle superfici piuttosto valido. In realtà rimane un piccolo punto da chiarire che potrebbe essere sicuramente un punto di svolta interessante per futuri lavori. Infatti più volte vi abbiamo definito la Photometric Stereo come una tecnica che consente di ricostruire delle superfici 3-D. Parlare però di ricostruzioni 3-D non è propriamente corretto. Anche perché le $z$ che abbiamo ottenuto sono delle funzioni in due variabili e non in tre come ci si aspetterebbe da un modello 3-D standard. Questo fatto avrete avuto modo di riscontrarlo anche precedentemente nei risultati che abbiamo riportato, in quanto non abbiamo una rappresentazione a tutto tondo degli oggetti. In ambito archeologico questo fenomeno è meno evidente quando si ha a che fare con dei rilievi. Ma se si volesse ricostruire un modello 3-D per esempio di un vaso o un'intera struttura preistorica la sola Photometric Stereo non sarebbe sufficiente. In molti di questi casi si ricorre alla strumentazione Laser, che però, come detto più volte, ha i suoi limiti che sono principalmente legati al costo e ai lunghi tempi di acquisizione. La domanda  che ci siamo posti, a fronte degli ottimi risultati ottenuti, è stata:

esiste un modo per integrare l'utilizzo della Photometric Stereo con qualcos' altro che consenta di ricostruire una superficie a 360°? La risposta è ovviamente si. Infatti si potrebbe pensare di ricorrere al Multiview, che in realtà abbiamo citato con un altro nome nel primo capitolo di questo testo (1.2 Fotometria binoculare). Pertanto si potrebbe pensare di ripetere il processo di acquisizione non da un solo punto di vista ma da diversi attorno alla superficie. Per ogni singola acquisizione si ricostruisce quindi un semi modello 3-D utilizzando le tecniche descritte in questo testo, e successivamente si crea una mappa di corrispondenza con tutti i modelli ottenuti. Questa operazione consentirebbe una rappresentazione 3-D a tutto tondo dei nostri oggetti. Dal punto di vista computazionale significherebbe moltiplicare il numero dei diversi punti di acquisizione per il tempo impiegato per ogni singola ricostruzione, in più ovviamente va sommato il tempo necessario per la costruzione della mappa delle corrispondenze, che sicuramente non è un'operazione banale. Siamo comunque abbastanza fiduciosi del fatto che si potrebbero ottenere risultati decisamente migliori rispetto a quelli che si otterrebbero con strumentazioni Laser, in un tempo decisamente inferiore, per non parlare della semplicità di acquisizione.

Ragionando sempre in ottica archeologica, un'altro sviluppo interessante sarebbe quello di realizzare un'applicazione completa dotata di interfaccia grafica. La motivazione è piuttosto evidente, per quanto infatti Matlab sia un ambiente di sviluppo semplice da utilizzare, richiede comunque delle conoscenze che non appartengono al bagaglio culturale di tutti gli archeologi. Motivo per il quale un'applicazione completa dotata di GUI grafica semplificherebbe molto il lavoro e non implicherebbe la presenza diretta degli ingegneri sul campo di lavoro. Ovviamente per rendere ancora più completo e funzionale il tutto, si potrebbe pensare di rendere l'app interfacciabile con dispositivi portatili come tablet e stampanti 3-D, in grado quindi di ricostruire le superfici precedentemente processate.

In conclusione possiamo affermare che tutti gli obbiettivi che ci siamo prefissati all'inizio di questo lungo e appassionate lavoro sono stati raggiunti. Il problema dovuto ai lunghi tempi di integrazione appartengono ormai al passato e questo sicuramente ci ha consentito di rilanciare una tecnica, che dai più è sempre stata scartata. Non solo, siamo stati in grado di costruire un algoritmo di elaborazione dei gradienti che non prevede l'utilizzo delle direzioni luce, e questo come abbiamo più volte fatto presente risulta essere un grosso vantaggio. Certo sono ancora tanti gli interrogativi e le migliorie che possono essere apportate, ma i risultati ottenuti fanno sicuramente ben sperare per i futuri sviluppi e aggiornamenti.

# Bibliografia


[1] G. Rodriguez, *Algoritmi Numerici*, Pitagora Editrice Bologna

[2] G. Rodriguez, S. Seatzu, *Introduzione alla Matematica Applicata e Computazionale*, Pitagora Editrice Bologna

[3] Redivo-Zaglia, Michela, and Giuseppe Rodriguez. *smt: a Matlab toolbox for structured matrices*. *Numerical Algorithms* 59.4 (2012): 639-659.

[4] G. Rodriguez, *Software: http://bugs.unica.it/~gppe/soft/*

[5] R. Pintus, M. Vanzi, *Advances in Photometric Sereo.*

[6] Harker, Matthew, and Paul O'Leary. *Least squares surface reconstruction from gradients: Direct algebraic methods with spectral, Tikhonov, and constrained regularization*. *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE, 2011.

[7] I. Rabascall, *Uncalibrated Photometric Stereo for 3D Surface Texture Recovery,* pages 4-25, 2003.

[8] Chen, Chia-Ping, and Chu-Song Chen. *The 4-source Photometric Stereo under general unknown lighting*. Computer Vision–ECCV 2006. Springer Berlin Heidelberg, 2006. 72-83.

[9] Basri, Ronen, David Jacobs, and Ira Kemelmacher. *Photometric Stereo with general, unknown lighting*. International Journal of Computer Vision 72.3 (2007): 239-257.

[10] Nagel, James R. *Solving the Generalized Poisson Equation Using the Finite-Difference Method (FDM*). *Lecture Notes, Dept. of Electrical and Computer Engineering, University of Utah* (2011).

[11] Å. Björck, *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia, 1996.

[12] L. Eldén, *Matrix Methods in Data Mining and Pattern Recognition*, SIAM, Philadelphia, 2007.

[13] C. Mannu, *Photometric Stereo for 3D Mapping of Carvings and Relieves*,2014.