UNIVERSITÀ DEGLI STUDI DI PADOVA
Facoltá di Scienze MM. FF. NN.
Dipartimento di Matematica Pura e Applicata
tesi di
Laurea Triennale in Matematica

Un algoritmo per la quadrangolazione convessa di poligoni

Relatore: Ch.mo Dott. Alvise Sommariva
Correlatore: Ch.mo Prof. Marco Vianello

Laureando: Mariano Gentile

a.a. 2009/2010

Sommario

Lo scopo di questo lavoro e' di descrivere un algoritmo di seguito implementato in Matlab che suddivida un qualsiasi poligono normale con n lati in m quadrilateri convessi e triangoli, in numero minore possibile. Daremo una stima per eccesso di m in funzione di n e del numero di angoli con ampiezza maggiore di π radianti, da comparare con la stima di algoritmi che dividano i poligoni in soli triangoli.

Indice

1 Algoritmo

1.1 Introduzione

Il proposito di questa sezione è di descrivere il funzionamento di un algoritmo, mostrato in dettaglio nelle successive pagine, e implementato in Matlab, che suddivida un poligono normale (cioè senza autointersezioni) in quadrilateri convessi e triangoli. Ovviamente il nostro obiettivo sarà di ridurre al minimo il numero dei sottopoligoni risultanti dalla suddivisione.
L'idea della suddivisione consiste nell'individuare un angolo $\mathbf{P}_{k-1}\widehat{\mathbf{P}_k\mathbf{P}}_{k+1}$ del poligono avente ampiezza maggiore di π radianti e nel tracciare all'interno del poligono normale iniziale $\mathcal{P}$ (che non è necessariamente convesso), un segmento, totalmente interno a $\mathcal{P}$, che abbia gli estremi sulla sua frontiera, e che non coincida con parte di essa, a meno degli estremi. In questo modo, dopo la prima suddivisione, avremo ottenuto due poligoni (neanche loro necessariamente convessi), $\mathcal{Q}$ e $\mathcal{S}$, i quali avranno in comune un lato, costituito dal segmento tracciato: in particolare essi saranno caratterizzati dal fatto che $\mathcal{P} = \mathcal{Q} \cup \mathcal{S}$ e inoltre $\mathring{\mathcal{Q}} \cap \mathring{\mathcal{S}} = \emptyset$, dove con $\mathring{\mathcal{Q}}$ e $\mathring{\mathcal{S}}$ intendiamo l'interno degli insiemi in questione (diremo usando una terminologia comune, che i poligoni $\mathcal{Q}$ e $\mathcal{S}$ non si *sovrappongono*). Quindi il problema si ridurrà a studiare il metodo migliore di tracciare tali segmenti. L'algoritmo è formato da due fasi: la prima fase controlla se il poligono immesso

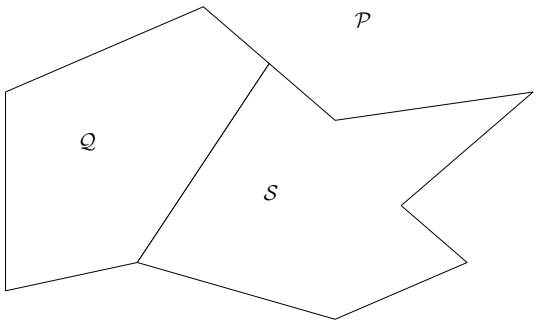

Figura 1: *Esempio di suddivisione di* $\mathcal{P}$

in input sia concavo, nel qual caso lo suddivide in poligoni convessi, mentre la seconda fase divide agevolmente i poligoni risultanti in quadrilateri e triangoli. Cominciamo con il descrivere quest'ultimo punto.

1.2 Suddivisione di poligoni convessi

L'algoritmo proposto per la suddivisione di poligoni convessi, consiste nel raggruppare di volta in volta quattro vertici consecutivi, determinando in questo modo un quadrilatero, che sarà convesso (dato che lo è tutto il poligono su cui stiamo lavorando). Dopo la prima suddivisione, avremo ricavato dal poligono iniziale $\mathcal{P}$ (supponiamolo di $n \geqslant 5$ vertici), un quadrilatero convesso e un altro poligono, ancora convesso, con $n - 2 \geqslant 3$ vertici. Da qui, iterando l'algoritmo, otterremo una serie di m quadrilateri $\mathcal{D}_i$, con $i = 1, \dots m$ dove al massimo uno solo di essi sarà un triangolo, fatto che accade solo nel caso che $\mathcal{P}$ abbia un numero dispari di vertici. Ovviamente per i $\mathcal{D}_i$ varranno le seguenti due proprietà :

$$\bigcup_{i=1}^{m} \mathcal{D}_i = \mathcal{P} \text{ e inoltre } \mathring{\mathcal{D}}_i \cap \mathring{\mathcal{D}}_j = \emptyset, \forall i \neq j$$

A questo punto dobbiamo quantificare il numero di quadrilateri m sopra citato.

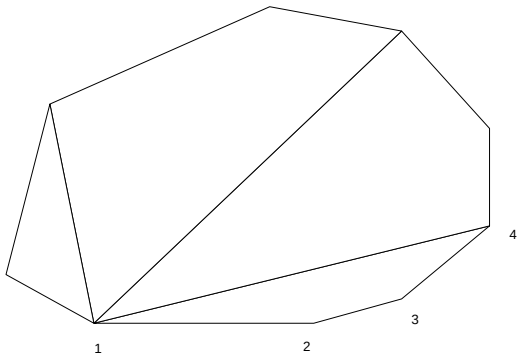

Figura 2: *Esempio di suddivisione di $\mathcal{P}$ convesso in quadrilateri con un triangolo di scarto*

Ricordiamo a questo proposito il significato della funzione *ceil*, in simboli $\lceil \cdot \rceil$: dato un numero reale, essa restituisce il più piccolo intero che sia maggiore o uguale del numero dato. Enunciamo quindi:

Teorema

Dato un poligono convesso, è possibile dividerlo in $\left\lceil \frac{n-2}{2} \right\rceil$ quadrilateri convessi non sovrapposti (di cui al massimo uno è un triangolo).

Dimostrazione

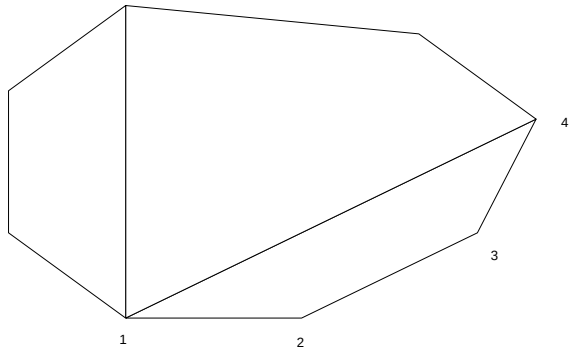

Figura 3: *Esempio di suddivisione di $\mathcal{P}$ convesso in quadrilateri*

Procediamo per induzione:

- *Triangolo*: $\lceil \frac{3-2}{2} \rceil = \lceil 1/2 \rceil = 1$
- *Quadrilatero convesso*: $\lceil \frac{4-2}{2} \rceil = \lceil 2/2 \rceil = 1$
- *Passo induttivo*: supponendo la validità per un poligono di n vertici, consideriamone uno di $n+2$ vertici, ovviamente con $n \geqslant 3$. Effettuando la divisione come sopra indicato, otteniamo un quadrilatero convesso più un poligono convesso di n vertici per il quale vale la formula perciò

$$1 + \left\lceil \frac{n-2}{2} \right\rceil = \left\lceil 1 + \frac{n-2}{2} \right\rceil = \left\lceil \frac{(n+2)-2}{2} \right\rceil$$

  avendo usato il fatto ovvio che possiamo portare dentro la funzione $\lceil \cdot \rceil$ numeri interi senza alterare il risultato (infatti essa agisce solo sulla parte frazionaria dei numeri, aumentandola fino all'unità se presente). $\square$

1.3 Funzionamento della prima fase dell'algoritmo

Vediamo ora in cosa consiste la prima fase dell'algoritmo che dato un poligono $\mathcal{P}$ concavo, fornisce una sequenza di poligoni convessi $\mathcal{P}_i$ non sovrapposti e tali che $\mathcal{P} = \bigcup_i \mathbf{P}_i$.
Come accennato nell'introduzione di questa sezione, l'idea consiste nel suddividere $\mathcal{P}$ in due poligoni $\mathcal{Q}$ e $\mathcal{S}$, e procedere successivamente per iterazione su ognuno di essi. Il punto chiave consiste nel determinare un segmento interno a $\mathcal{P}$ che unisca due punti sulla sua frontiera, chiamati $\mathbf{A}$ e $\mathbf{B}$. La buona scelta di

questi punti sarà fondamentale nella costruzione dell'algoritmo.
Consideriamo quindi $\mathcal{P}$ avente tra i vertici almeno un angolo *concavo*, ossia strettamente maggiore di π radianti, sia esso compreso tra i vertici $\mathbf{P}_{k-1}\widehat{\mathbf{P}_k\mathbf{P}_{k+1}}$. Si osservi che la proprietà di un poligono di non essere convesso coincide con la proprietà di possedere almeno un angolo concavo e saranno proprio questi angoli che cercheremo di eliminare nelle suddivisioni sotto esposte.
Una scelta possibile per il segmento $\overline{\mathbf{AB}}$ è di prendere $\mathbf{A} = \mathbf{P}_k$, prolungare il segmento $\overline{\mathbf{P}_{k-1}\mathbf{P}_k}$, e successivamente prendere come $\mathbf{B}$ la prima intersezione tra questo prolungamento e la frontiera di $\mathcal{P}$. Dovendo essere il segmento interno a $\mathcal{P}$, considereremo il prolungamento dalla parte di $\mathbf{P}_k$ (si osservi la figura), che incontrerà sicuramente una intersezione con la frontiera di $\mathcal{P}$ essendo questa una curva chiusa.
Osserviamo che così facendo abbiamo in effetti creato due poligoni $\mathcal{Q}$ e $\mathcal{S}$, non sovrapposti, la cui unione è $\mathcal{P}$.
Si osservi che è di fondamentale importanza che con la suddivisione effettuata

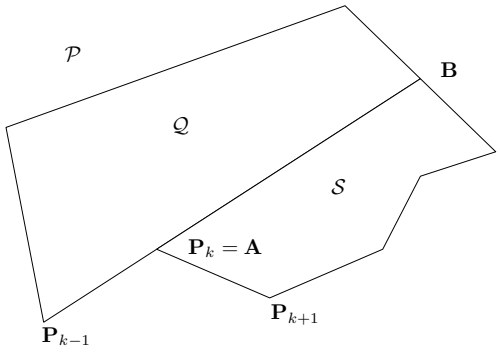

Figura 4: *Esempio di suddivisione*

l'angolo concavo $\mathbf{P}_{k-1}\widehat{\mathbf{P}_k\mathbf{P}_{k+1}}$ resta diviso in due angoli di ampiezza $\leqslant \pi$ radianti ed in particolare un angolo esattamente piatto, ossia $\mathbf{P}_{k-1}\widehat{\mathbf{P}_k\mathbf{B}} = \pi$ radianti. L'allineamento di questi tre punti giocherà un ruolo fondamentale nel definire successivamente una stima superiore. Riguardo il restante angolo, notiamo invece che $\mathbf{B}\widehat{\mathbf{P}_k\mathbf{P}_{k+1}} < \pi$, ove questa disuguaglianza è stretta.
Vediamo cosa succede invece dalla parte opposta del segmento, nell vicinanze di $\mathbf{B}$.
Come già detto, $\mathbf{B}$ è stato scelto come intersezione del prolungamento di $\overline{\mathbf{P}_{k-1}\mathbf{P}_k}$ con la frontiera di $\mathcal{P}$. Avremo quindi due possibilità :

- $\mathbf{B} = \mathbf{P}_j + t(\mathbf{P}_{j+1} - \mathbf{P}_j)$ con $t \in (0,1)$, in particolare $t \neq 0, 1$, per qualche vertice indicizzato con j, ossia $\mathbf{B}$ interno ad un lato di $\mathcal{P}$.
- $\mathbf{B} = \mathbf{P}_j$ per un qualche indice j, ossia $\mathbf{B}$ coincide con un vertice di $\mathcal{P}$ già esistente.

Studiamo in dettaglio questi due casi.

- **B interno ad un lato di $\mathcal{P}$.**
  In questo caso l'intersezione sopra costruita, provoca la creazione di due angoli, $\widehat{\mathbf{P}_{j+1}\mathbf{B}\mathbf{A}}$ e $\widehat{\mathbf{P}_j\mathbf{B}\mathbf{A}}$. Non è difficile convincersi che entrambi questi

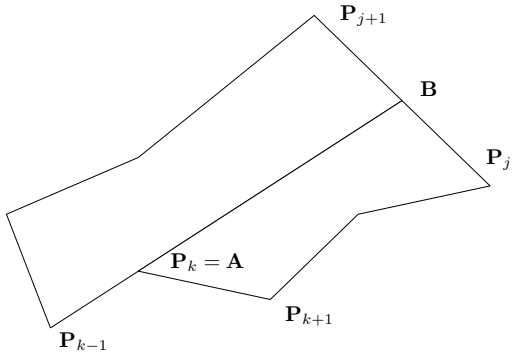

Figura 5: $\mathbf{B}$ *interno ad un lato di* $\mathcal{P}$.

  angoli devono essere strettamente minori di π radianti, dato che la loro somma deve essere esattamente uguale a π, ed inoltre, per costruzione, entrambi saranno interni al poligono $\mathcal{P}$.
  Quindi in questa prima eventualità , notiamo soprattutto che l'angolo concavo da cui eravamo partiti $\widehat{\mathbf{P}_{k-1}\mathbf{P}_k\mathbf{P}_{k+1}}$ è stato diviso in due angoli convessi di cui uno in particolare è piatto per costruzione, e dei due angoli creati ex-novo, nessuno è concavo.

- **B coincide con un vertice di $\mathcal{P}$.**
  Come anticipato, la seconda eventualità è che sia $\mathbf{B} = \mathbf{P}_j$, quindi il segmento $\overline{\mathbf{AB}}$ dividerà in due l'angolo $\widehat{\mathbf{P}_{j-1}\mathbf{P}_j\mathbf{P}_{j+1}}$, formato da tre vertici già esistenti in $\mathcal{P}$. Si possono ora verificare quattro differenti casi, come mostrato nelle figure sottostanti.

1. **B divide un angolo convesso.**
   Questa è l'eventualità più semplice, e non abbiamo molto da verifi-

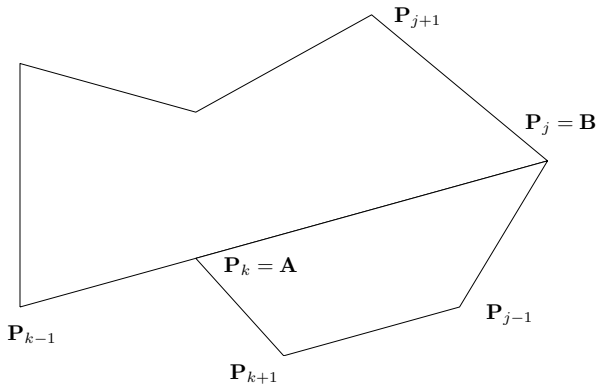

Figura 6: **B** *divide un angolo convesso.*

   care: un angolo già convesso di $\mathcal{P}$ viene ulteriormente diviso in due angoli, che saranno ovviamente ancora convessi.

2. **B divide un angolo concavo, e successivamente alla divisione, uno dei due angoli resta concavo.**
   In questo caso, $\widehat{\mathbf{P}_{j-1}\mathbf{P}_j\mathbf{P}_{j+1}}$ è stato diviso in $\widehat{\mathbf{P}_{j-1}\mathbf{B}\mathbf{A}}$, che (come

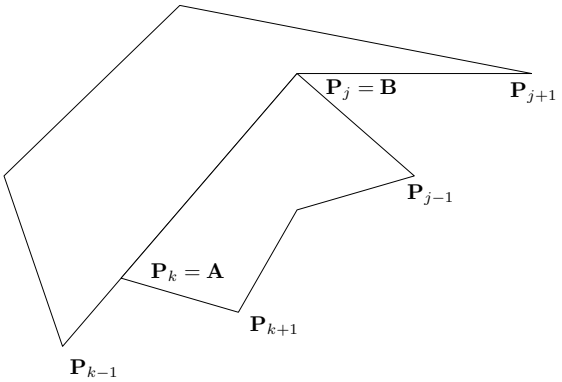

Figura 7: **B** *divide un angolo concavo, e successivamente alla divisione, uno dei due angoli resta concavo.*

   in figura) è convesso, e $\widehat{\mathbf{A}\mathbf{B}\mathbf{P}_{j+1}} = \widehat{\mathbf{P}_{j-1}\mathbf{P}_j\mathbf{P}_{j+1}} - \widehat{\mathbf{P}_{j-1}\mathbf{B}\mathbf{A}}$, che supponiamo essere ancora concavo. Quest'ultimo sarà quindi diviso nuo-

vamente nella successiva iterazione dell'algoritmo. Cosa ovvia ma importante da notare nonchè utile in seguito, volendo sommare il numero di angoli concavi di $\mathcal{P}$ dopo la divisione, essi non sono aumentati anzi sono diminuiti di una unità : nel conteggio l'angolo concavo $\widehat{\mathbf{ABP}}_{j+1}$ va a sostituire $\mathbf{P}_{j-1}\widehat{\mathbf{P}_j\mathbf{P}}_{j+1}$ che era concavo prima della divisione.

3. **B divide un angolo concavo, dividendolo in due angoli convessi.**

   Questa eventualità è particolarmente favorevole, in quanto siamo

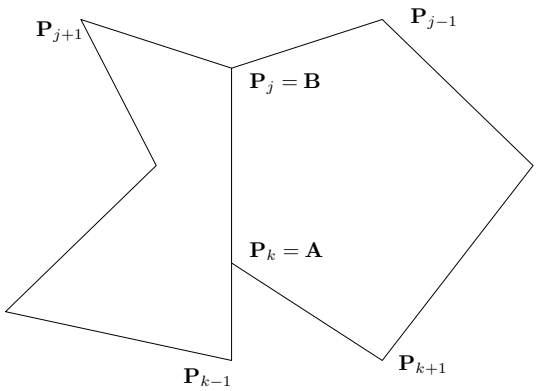

Figura 8: **B** *divide un angolo concavo, dividendolo in due angoli convessi.*

   riusciti ad eliminare due angoli concavi con una sola iterazione della prima fase dell'algoritmo. Si vedrà successivamente, ma qui lo accenniamo, che l'algoritmo dovrà essere iterato ogni qualvolta sia presente un angolo concavo in uno dei sottopoligoni risultanti dalla suddivisione, quindi l'essere riusciti a dividere due angoli concavi in angoli convessi in un colpo solo, corrisponde a dover iterare l'algoritmo una volta di meno rispetto al caso generale.

4. **B divide un angolo concavo, dividendolo in due angoli, uno convesso e uno piatto.**

   Questa è un caso particolare del punto precedente: anche qui possiamo iterare l'algoritmo una volta di meno, ma in più adesso, il segmento $\overline{\mathbf{AB}}$, già prolungamento di $\overline{\mathbf{P}_{k-1}\mathbf{P}_k}$, si prolunga in $\overline{\mathbf{P}_j\mathbf{P}_{j+1}}$ (si guardi la figura).
   Rispetto al caso precedente, la novità qui è il poter allineare i tre punti $\overline{\mathbf{P}_k\mathbf{P}_j\mathbf{P}_{j+1}}$, e questo in casi particolari potrebbe ridurre la stima superiore del numero di quadrangoli prodotti dall'algoritmo. Vedremo in seguito ulteriori dettagli di tale questione.

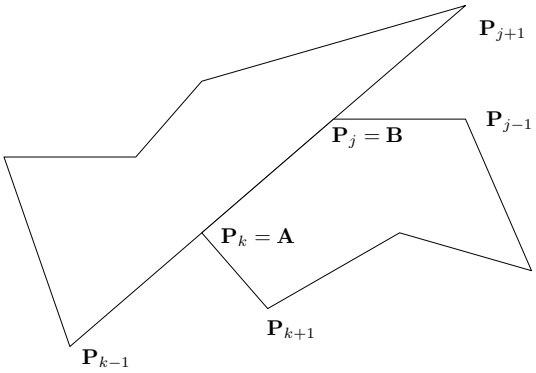

Figura 9: **B** *divide un angolo concavo, dividendolo in due angoli, uno convesso e uno piatto.*

1.4 Stima del numero di poligoni convessi ottenuti dall'algoritmo.

Dopo aver analizzato come effettuare una singola suddivisione ed aver visto come posizionare il segmento $\overline{\mathbf{AB}}$, occupiamoci del numero di iterazioni necessarie alla prima fase dell'algoritmo per ottenere una suddivisione composta esclusivamente di poligoni convessi. Sia quindi $\mathcal{P}$ un poligono avente n vertici di cui k siano concavi, ossia strettamente maggiori di π radianti. Chiamiamo $|\mathcal{P}|_{conc}$ il numero di angoli concavi di $\mathcal{P}$, quindi $|\mathcal{P}|_{conc} = k$. Alla J-esima iterazione dell'algoritmo sia $\mathcal{P} = \bigcup_{i=1}^{J} \mathcal{P}_i$ con $\bigcap_{i=1}^{J} \mathring{\mathcal{P}}_i = \emptyset$ e supponiamo che $\mathcal{P}_l$ con $l \leqslant J$ sia non convesso e quindi da suddividere ulteriormente. Per come è definito l'algoritmo otterremo due poligoni $\mathcal{Q}_l$ e $\mathcal{S}_l$ con la proprietè che $|\mathcal{Q}_l|_{conc} + |\mathcal{S}_l|_{conc} < |\mathcal{P}_l|_{conc}$, ove la disuguaglianza è stretta in quanto ad ogni iterazione andiamo a dividere esattamente un angolo concavo in due convessi (anzi, in alcuni casi particolari sopra visti, a volte riusciamo a dividere anche due angoli concavi per iterazione, ma essendo questa un'eventualità del tutto fortuita ed incontrollabile, non ce ne occuperemo, limitandoci a notare la sua esistenza), in particolare nella suddivisione non abbiamo aggiunto nessun angolo concavo ex-novo.
Quindi, ricordando che k è il numero di angoli concavi del poligono $\mathcal{P}$, dopo $m \leqslant k$ iterazioni avremo $m+1$ poligoni $\mathcal{P}_1, \ldots, \mathcal{P}_{m+1}$ tali che:

$$\mathcal{P} = \bigcup_{i=1}^{m+1} \mathcal{P}_i$$

$$\mathring{\mathcal{P}}_i \cap \mathring{\mathcal{P}}_j = \emptyset, \forall i \neq j$$

$$\sum_{i=1}^{m+1} |\mathcal{P}_i|_{conc} \leqslant k - m$$

Le prime due proprietà sono state già viste, mentre l'ultima dice che la prima fase dell'algoritmo terminerà in al più k iterazioni, non potendo il numero di angoli concavi essere negativo. Supponiamo quindi che tale numero di iterazioni sia $\overline{m}$, avremo quindi $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{\overline{m}+1}$ poligoni convessi (infatti ogni divisione aumenta il numero totale di poligoni di una unità). A questo punto, possiamo applicare ad ogni $\mathcal{P}_i$ la seconda fase dell'algoritmo e otterremo un numero di quadrilateri, che non si sovrappongono e la cui unione sia $\mathcal{P}$:

$$\left\lceil \frac{n_1 - 2}{2} \right\rceil + \cdots + \left\lceil \frac{n_{\overline{m}+1} - 2}{2} \right\rceil$$

ove $n_1, \ldots, n_{\overline{m}+1} \geqslant 3$ rappresentano il numero di vertici dei poligoni $\mathcal{P}_1, \ldots, \mathcal{P}_{\overline{m}+1}$, e di questi quadrilateri convessi al massimo $\overline{m} + 1$ sono triangoli.

1.5 Stima superiore del numero di quadrilateri risultanti.

Nella precedente sezione abbiamo visto quale sarà il numero di quadrilateri risultanti, e quanti di loro possono essere triangoli, avendo supposto di conoscere il numero di angoli concavi k, il numero di iterazioni $\overline{m} \leqslant k$ e il numero di vertici di ogni poligono convesso creato durante la prima fase. Supporremo ancora di conoscere il numero di angoli concavi k di $\mathcal{P}$, cosa sempre possibile, visto che siamo noi a definire $\mathcal{P}$.
Per fare ulteriori stime analizziamo quindi alcune proprietà della funzione $\lceil \cdot \rceil$.

1.5.1 Funzione *ceil*

Ci limitiamo ad analizzare le proprietà che interessano all'algoritmo in questione, in particolare come si comporta la funzione con numeri che possono essere interi oppure avere 0.5 come parte decimale. Questo poichè il numero di vertici deve essere sempre un numero intero, e quindi le uniche possibilità su cui dovremo riflettere saranno quelle in cui i vertici sono in numero pari o dispari, dato che nella funzione $\lceil \cdot \rceil$ è presente una divisione per due.

Cominciamo prendendo due numeri e studiando il comportamento di $\lceil \cdot \rceil$ nei vari casi possibili.

1. n_1 e n_2 sono numeri pari, allora anche la loro somma è pari, il che significa

che $n_1/2$, $n_2/2$ e anche $(n_1+n_2)/2$ sono interi perciò

$$\left\lceil \frac{n_1}{2} \right\rceil + \left\lceil \frac{n_2}{2} \right\rceil = \frac{n_1}{2} + \frac{n_2}{2} = \left\lceil \frac{n_1+n_2}{2} \right\rceil$$

osservando che la funzione $\lceil \cdot \rceil$ applicata ad un numero intero, è il numero stesso.

2. n_1 è un numero pari mentre n_2 è dispari, ciò significa che anche la loro somma è dispari, e in particolare $n_2/2$ e $(n_1+n_2)/2$ non sono numeri interi. Allora otteniamo

$$\left\lceil \frac{n_1}{2} \right\rceil + \left\lceil \frac{n_2}{2} \right\rceil = \frac{n_1}{2} + \left\lceil \frac{n_2}{2} \right\rceil = \left\lceil \frac{n_1}{2} + \frac{n_2}{2} \right\rceil =$$

$$= \left\lceil \frac{n_1+n_2}{2} \right\rceil = \frac{n_1+n_2}{2} + 0.5$$

osservando che possiamo sommare o sottrarre numeri interi dentro la funzione $\lceil \cdot \rceil$.

3. n_1 e n_2 sono entrambi numeri dispari, quindi $n_1/2$ e $n_2/2$ non sono interi, mentre la loro somma è pari, quindi $(n_1+n_2)/2$ è un intero. Si vede subito che

$$\left\lceil \frac{n_1}{2} \right\rceil + \left\lceil \frac{n_2}{2} \right\rceil = \frac{n_1}{2} + 0.5 + \frac{n_2}{2} + 0.5$$

$$\left\lceil \frac{n_1+n_2}{2} \right\rceil = \frac{n_1+n_2}{2}$$

e quindi

$$\left\lceil \frac{n_1}{2} \right\rceil + \left\lceil \frac{n_2}{2} \right\rceil = \left\lceil \frac{n_1+n_2}{2} \right\rceil + 1$$

1.5.2 Stima superiore.

Calcoliamo ora una stima del numero di poligoni convessi, ricordando che il loro numero è al più $\overline{m}+1 \leqslant k+1$, e sfruttando le proprietà appena descritte della funzione $\lceil \cdot \rceil$. Consideriamo il caso meno fortunato in cui $\overline{m}+1 = k+1$. Il numero da stimare è diventato:

$$\left\lceil \frac{n_1-2}{2} \right\rceil + \cdots + \left\lceil \frac{n_{\overline{m}+1}-2}{2} \right\rceil \leqslant$$

$$\left\lceil \frac{n_1-2}{2} \right\rceil + \cdots + \left\lceil \frac{n_{k+1}-2}{2} \right\rceil$$

A questo punto dovremo mettere in relazione i vari n_i con n che conosciamo. Cominciamo notando che:

$$\left\lceil \frac{n_1 - 2}{2} \right\rceil + \cdots + \left\lceil \frac{n_{k+1} - 2}{2} \right\rceil =$$

$$\left\lceil \frac{n_1}{2} \right\rceil - 1 + \cdots + \left\lceil \frac{n_{k+1}}{2} \right\rceil - 1 =$$

$$\left\lceil \frac{n_1}{2} \right\rceil + \cdots + \left\lceil \frac{n_{k+1}}{2} \right\rceil - (k+1)$$

Ora dalle osservazioni viste riguardo alla funzione $\lceil \cdot \rceil$ possiamo dire ancora:

$$\left\lceil \frac{n_1}{2} \right\rceil + \cdots + \left\lceil \frac{n_{k+1}}{2} \right\rceil - (k+1) \leqslant$$

$$\left\lceil \frac{n_1 + \cdots + n_{k+1}}{2} \right\rceil + \left\lfloor \frac{k+1}{2} \right\rfloor - (k+1)$$

dove ricordiamo che la funzione $floor$, in simboli $\lfloor \cdot \rfloor$, restituisce il maggior intero minore o uguale del numero in questione. Infatti la funzione *ceil* aumenta di una unità solo nel caso in cui si sommino una coppia di numeri dispari al suo interno, questo spiega l'utilizzo di $\left\lfloor \frac{k+1}{2} \right\rfloor$.
A questo punto l'unica cosa che ci rimane da analizzare è la sommatoria $\sum_{i=1}^{k+1} n_i$. Per far questo ripercorriamo la costruzione del segmento $\overline{\mathbf{AB}}$: esso è il prolungamento del segmento $\overline{\mathbf{P}_{k-1}\mathbf{P}_k}$, che va ad intersecare la frontiera di $\mathcal{P}$ in un vertice esistente, o all'interno di un suo lato, creando così un nuovo vertice. Guardiamo le varie possibilità :

1. $\mathbf{B}$ coincide con un vertice, sia esso $\mathbf{P}_j$. Allora i due poligoni, chiamiamoli $\mathcal{Q}$ e $\mathcal{S}$, risultanti da questa iterazione, avranno in comune $\mathbf{A} = \mathbf{P}_k$ (si ricordino le figure sopra esposte) e $\mathbf{B} = \mathbf{P}_j$, e tutti gli altri vertici da loro posseduti, sono gli stessi di $\mathcal{P}$. Questi ultimi erano contati una volta nel numero n di vertici di $\mathcal{P}$, e continueranno ad essere contati una sola volta nella sommatoria dei vertici di $\mathcal{Q}$ e $\mathcal{S}$, sia essa $n_Q + n_S$. $\mathbf{B}$ è ora contato due volte invece che una, mentre $\mathbf{A}$ viene contato ancora solamente una volta: è questo il motivo della particolare costruzione del segmento $\overline{\mathbf{AB}}$ come prolungamento di $\overline{\mathbf{P}_{k-1}\mathbf{P}_k}$, infatti ora in uno dei due poligoni risultanti, precisamente quello contente il vertice $\mathbf{P}_{k-1}$, avremo tre punti $\overline{\mathbf{P}_{k-1}\mathbf{P}_k\mathbf{B}}$ allineati, dei quali possiamo evitare di considerare quello centrale. A questo punto, è ovvio che risulta $n_Q + n_S = n + 1$.

2. $\mathbf{B}$ non coincide con nessun vertice di $\mathcal{P}$. Quindi esso non compare nel numero n, mentre sarà contato due volte in $n_Q + n_S$. Per $\mathbf{P}_k$ invece il ragionamento è lo stesso del punto precedente, quindi in questo caso risulta $n_Q + n_S = n + 2$.

Possiamo quindi affermare che dopo ogni iterazione avremo $n_Q + n_S \leqslant n + 2$, quindi alla fine risulterà $\sum_{i=1}^{k+1} n_i \leqslant n + 2k$. Riprendendo la formula possiamo finalmente trovare la stima superiore del numero di quadrilateri convessi risultanti dall'esecuzione dell'algoritmo su $\mathcal{P}$, conoscendone il numero di vertici n e il numero di angoli concavi k, e possiamo anche affermare che al massimo $k+1$ di questi quadrilateri saranno in realtà triangoli:

$$\left\lceil \frac{n_1 + \cdots + n_{k+1}}{2} \right\rceil + \left\lfloor \frac{k+1}{2} \right\rfloor - (k+1) \leqslant$$

$$\left\lceil \frac{n+2k}{2} \right\rceil + \left\lfloor \frac{k+1}{2} \right\rfloor - (k+1) =$$

$$\left\lceil \frac{n}{2} \right\rceil + k + \left\lfloor \frac{k+1}{2} \right\rfloor - (k+1) =$$

$$\left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{k+1}{2} \right\rfloor - 1$$

1.5.3 Paragone con la triangolazione.

Paragoniamo l'algoritmo fin qui esposto con la triangolazione, ossia la divisione di un poligono in soli triangoli. Sappiamo che quest'ultima divide $\mathcal{P}$ di n vertici in $n-2$ triangoli, contro la nostra stima superiore

$$\left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{k+1}{2} \right\rfloor - 1$$

Quindi i due algoritmi danno risultati pressocchè uguali quando il numero di angoli concavi k è molto elevato (notiamo infatti che è possibile costruire poligoni con infiniti vertici in cui $k \approx n$, si veda sotto un esempio), mentre si ha grande vantaggio quando si riduce k.
Dalle considerazioni sopra fatte si nota inoltre che non solo il numero ma anche la disposizione degli angoli concavi ha il suo peso, e potrebbe ridurre anche notevolmente la stima sopra data.

1.6 Esempi

Diamo un esempio: scriviamo input e output risultante in Matlab per ottenere il poligono in Figura 13.

```
c=[4,7;6,10;8,8;10,10;11,8;9,7;8,5;9,3;11,2;15,6;15,10;18,10;19,12;17,15;15,15;12,12;8,13;10,16;...
6,16;3,14;2,11;3,8]
```

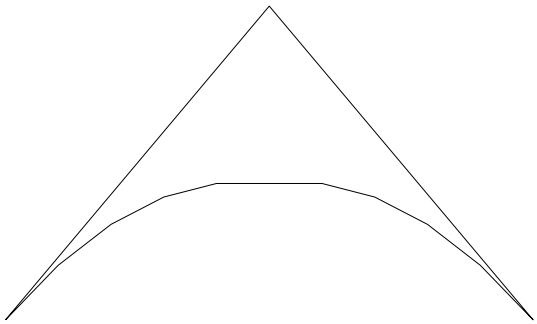

Figura 10: *Esempio di figura con 11 vertici di cui 8 concavi. Seguendo la procedura usata per creare questa figura, si capisce come poter costruire poligoni con un numero qualsiasi di vertici in cui* $k \approx n$.

Il vettore c è un vettore colonna contenente i vertici, il cui primo elemento non è ripetuto alla fine. I vertici del poligono sono scritti in senso antiorario.

```
[A,b]=divconcavitot(c,1)

A =

    10    16    10    16    12    12    11     8    11     8    15    10    15    10    15    15
     6    16     2    11     8    13     9     7     9     3    18    10    17    15    10    10
     3    14     3     8     6    10     8     5    11     2    19    12    15    15    13     4
     2    11     4     7     8     8     9     3    13     4    17    15     0     0    15     6


b =

     4     4     4     4     4     4     3     4
```

Nel richiamare la funzione abbiamo dato come input il vettore e una variabile binaria: il numero 1 disegna una figura del poligono suddiviso, 0 non la disegna. Nell'output notiamo una matrice A nelle cui colonne sono scritte le coordinate dei quadrangoli, nella forma

```
[x(1),y(1);x(2),y(2);...]
```

dove le x(i) e y(i) sono vettori colonna di dimensione 4.
Notiamo che nella penultima coppia di coordinate sono presenti degli 0: essi potrebbero significare un quadrangolo con un vertice nell'origine oppure un triangolo. Il vettore b chiarisce tale dubbio, essendo presente un 3 nella penultima posizione, sappiamo che quelle coordinate devono essere tralasciate, in quanto in quelle colonne sono presenti le coordinate di un triangolo, e gli 0 hanno solo la funzione di riempire la matrice.

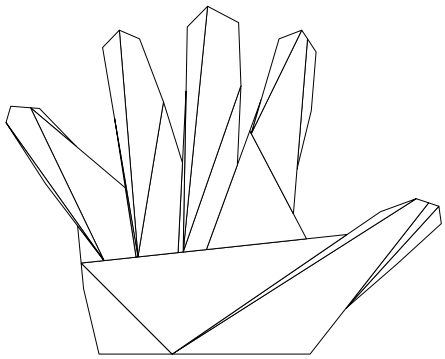

Figura 11: *Esempio di un poligono a forma di mano con 37 vertici. Dalla suddivisione risultano 24 quadrangoli, contro una possibile triangolazione che darebbe luogo a 35 triangoli.*

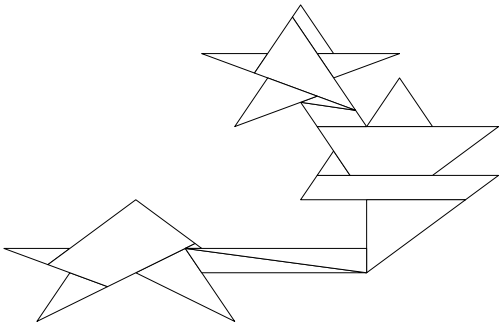

Figura 12: *Esempio di un poligono* ***strano*** *con 29 vertici. I quadrilateri risultanti sono 19 contro i 27 della triangolazione.*

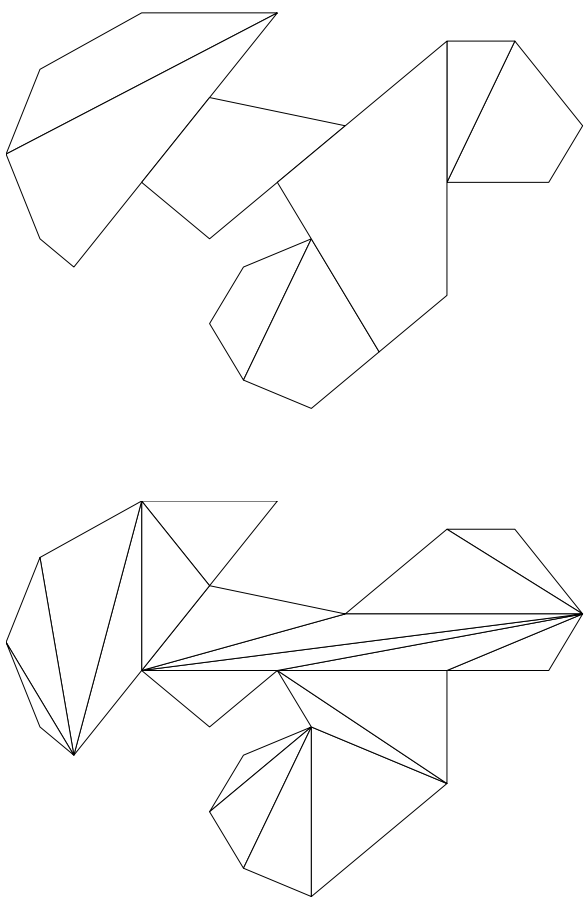

Figura 13: *Esempio di un poligono con un numero limitato di triangoli risultanti. Abbiamo 22 vertici, e l'algoritmo qui presentato suddivide il poligono in 8 quadrangoli, mentre nella figura in basso possiamo vedere una triangolazione sullo stesso poligono che produce 20 triangoli.*

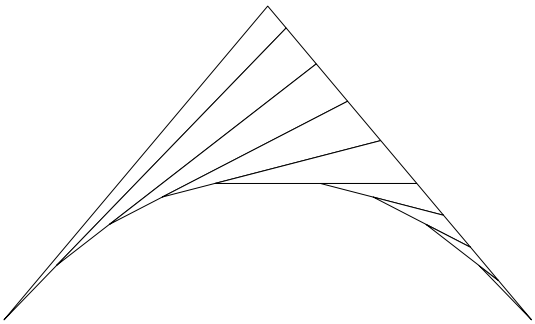

Figura 14: *Si noti in questa figura che $k \approx n$, in particolare non è possibile trovare nessun quadrilatero convesso nella suddivisione, e l'algoritmo sopra esposto si riduce ad una particolare triangolazione. Abbiamo infatti 11 vertici e 9 triangoli.*

1.7 Complessità

Diamo ora una stima circa la complessità dell'algoritmo. Il codice dell'algoritmo è presente alla fine di queste pagine.

Cominciando ad analizzare la funzione principale *divconcavitot.m* vediamo che nella prima parte, l'unica cosa interessante da analizzare è il richiamo a *divconcavi.m*, mentre il resto sono semplici scritture di matrici in modo da riorganizzare gli input successivi. La funzione *divconcavi.m* verrà richiamata fin quando ci sarà un poligono concavo da suddividere, e una volta per ogni poligono convesso trovato (infatti l'algoritmo riconosce un poligono convesso dall'output della funzione *divconcavi.m*). Dalle osservazioni sopra fatte, sappiamo che il numero di poligoni convessi sarà al massimo $k + 1$ e che il numero di richiami crescerà linearmente con k ove come sempre k è il numero di angoli concavi del poligono $\mathcal{P}$. Successivamente verrà richiamata la funzione *divpolyconvex.m* al massimo $k+1$ volte, per ognuno dei sottopoligoni convessi. Questa funzione è di complessità trascurabile, si limita infatti a raggruppare i vertici a gruppi di 4, e quindi a riorganizzare la matrice in output. Riassumendo dovremo considerare la complessità di $\mathcal{O}(k)$ richiami della funzione *divconcavi.m*.

La funzione *divconcavi.m* cerca il primo angolo concavo e prolungando il suo primo lato, cerca le intersezioni con gli altri lati del poligono per poi sceglierne una secondo alcuni criteri. Il procedimento verrà eseguito n volte, una per ogni vertice di $\mathcal{P}$ e per ogni iterazione, l'operazione più complessa è un sistema lineare

di matrice 2x2 (anche se questa è una situazione molto pessimista). Il resto di questa parte di algoritmo riorganizza l'output.

Concludendo possiamo affermare che la complessità dell'algoritmo sarà $\mathcal{O}(kn)$.

2 Interpolazione polinomiale e cubatura su poligoni

2.1 Introduzione: interpolazione di una funzione

Supponiamo di voler approssimare una funzione continua $f : \Omega \subset \mathbb{R}^d \to \mathbb{R}$ con un polinomio che interpoli f su un set di punti $x_1, x_2, \ldots, x_M \in \Omega$ ove Ω è un dominio compatto di $\mathbb{R}^d$. Tale problema consiste nel trovare un polinomio $p \in \mathbb{P}_n$ di grado n nelle d variabili tale che

$$p(x_i) = f_i, \forall i = 1, 2, \ldots, M$$

$$f_i = f(x_i), M = \binom{n+d}{d}$$

Per prima cosa si desidera che il problema sia ben posto, ossia che qualsiasi siano i valori f_i, $i = 1, \ldots, M$ la soluzione esista e sia unica: se $\phi_1, \ldots, \phi_M$ è una base di $\mathbb{P}_n$ e V_n la matrice di Vandermonde $V_n = V_n(x_1, \ldots, x_M) = [\phi_j(x_i)]$, allora ciò è vero se e solo se $\det(V_n) \neq 0$ in particolare le precedenti equazioni devono avere rango $M := \binom{n+d}{d} = dim(\mathbb{P}_n)$, ovvero impongano M equazioni linearmente indipendenti. Un tale insieme di punti A_n si dice unisolvente di grado n (in dimensione d) ed ha la proprietà che due polinomi di grado n in d variabili i cui valori coincidano su A_n, necessariamente coincidono in $\mathbb{P}_n$.
Come nel caso unidimensionale il polinomio interpolatore può essere calcolato facilmente una volta noti i polinomi di Lagrange

$$L_i(x) = \frac{\det(V(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_M))}{\det(V(x_1, \ldots, x_M))}$$

in quanto da $L_i(x_j) = \delta_{ij}$, si ha come nel caso unidimensionale che il polinomio interpolante $p \in \mathbb{P}_n$ è

$$p(x) = \sum_{i=1}^{M} f_i L_i(x)$$

Definita inoltre la costante di Lebesgue $\Lambda(\xi) := \max_{x\in\Omega} \sum_{i=1}^{M} |L_i(x)|$, dove ξ è l'insieme dei punti di interpolazione e i polinomi di Lagrange sono calcolati rispetto a ξ, si dimostra che se f è una funzione continua in Ω allora

$$\|f(x) - p(x)\|_\Omega \leq (1 + \Lambda(\xi))E(f)_n$$

dove $E(f)_n = dist(f(x), \mathbb{P}_n)$, distanza minima tra f e lo spazio polinomiale $\mathbb{P}_n$ nel dominio Ω.
Da questa disuguaglianza appare ovvia l'importanza della ricerca di una mesh

di punti che assicuri una crescita lenta della costante di Lebesgue al crescere del grado n del polinomio interpolante.
Purtroppo cercare dei punti che minimizzino la costante di Lebesgue è un problema computazionalmente molto complesso. Ci limitiamo quindi alla ricerca di punti che massimizzino il determinante $\det(V_n)$ della matrice di Vandermonde, i cosiddetti *punti di Fekete*: massimizzando $\det(V_n)$ nella definizione dei polinomi L_i, e osservando la definizione della costante di Lebesgue appare ovvia la seguente maggiorazione:

$$\Lambda_(\xi) := \max_{x\in\Omega} \sum_{i=1}^{M} |L_i(x)| = \max_{x\in\Omega} \sum_{i=1}^{M} \left| \frac{\det(V_n(x_1,\ldots,x_i-1,x,x_i+1,\ldots,x_M))}{\det(V_n(x_1,\ldots,x_M))} \right| \leq M$$

Altri punti notevoli sono i *punti de Leja* definiti con la procedura seguente: dato $K \subseteq \mathbb{C}^d$ insieme compatto continuo o discreto, supponiamo di avere una base *ordinata* di $\mathbb{P}_n$, sia ancora $\phi_1,\ldots,\phi_M$, ove $M := \binom{n+d}{n}$, i punti di Leja sono definiti da una sequenza $\xi_1,\ldots,\xi_M \in K$. Il primo punto ξ_1 è scelto in modo da massimizzare il primo polinomio della base, sia $\phi_1(x)$, in K (ci potrebbero essere più punti che massimizzano tale polinomio; in tal caso la scelta è indifferente, e ognuno definisce una differente sequenza di punti; questo ragionamento vale anche per i successivi ξ_i). Dati quindi $\xi_1,\ldots,\xi_k$ punti di Leja, il k+1-esimo punto ξ_{k+1} è definito in modo da massimizzare la funzione

$$x \longrightarrow |V(\xi_1,\ldots,\xi_k,x;\phi_1,\ldots,\phi_k,\phi_{k+1})|$$

ove $V(\xi,\Phi)$ rappresenta la matrice di Vandermonde calcolata nei punti ξ_i e nella base $\phi_1,\ldots,\phi_M$.

2.2 WAM - Weakly Admissible Meshes

Il calcolo di punti di Fekete e di Leja in un insieme compatto $\Omega \in \mathbb{R}^d$ è un problema ad alto costo computazionale, ed entrambi sono stati calcolati in pochi e semplici casi, come l'intervallo o il cerchio complesso nel caso univariato e il cubo e il cilindro nel caso multivariato.
D'altra parte recentemente è stata sviluppata la teoria delle *mesh ammissibili* (4) e delle mesh debolmente ammissibili (in inglese Weakly Admissible Mesh abbreviato con WAM) da cui è stato possibile estrarre i cosiddetti *punti approssimati di Fekete* e i *punti di Leja discreti*.

Definizione

Dato un insieme compatto $\Omega \subset \mathbb{R}^d$ (la definizione vale anche nel caso $\Omega \subset \mathbb{C}^d$), che sia determinante per i polinomi, ossia ogni polinomio che si annulla in Ω è il

polinomio identicamente nullo, una *Weakly Admissible Mesh* è definita come una sequenza di sottoinsiemi discreti $A_n \subset \Omega$ tali che per ogni numero naturale n

$$\|p\|_\Omega \leq C(A_n)\|p\|_{A_n}, p \in \mathbb{P}_n$$

dove sia la cardinalità di A_n ($\geq M$), sia la costante $C(A_n)$ (indipendente da p) crescono al più *polinomialmente* con n. Nel caso $C(A_n)$ sia limitata, si parla di Admissible Mesh.

Le WAMs godono di alcune importanti proprietà (3) che elenchiamo di seguito

1. $C(A_n)$ è invariante sotto trasformazioni affini;
2. una unione finita di WAMs è una WAM per la corrispondente unione di compatti su cui sono definite, avente come costante $C(A_n)$ il massimo tra le costanti;
3. data una mappa polinomiale π_s di grado s, allora $\pi_s(A_{ns})$ è una WAM per $\pi_s(\Omega)$ con costante $C(A_{ns})$;
4. la costante di Lebesgue dei punti di Fekete estratti da una WAM è limitata da $\Lambda_n \leq MC(A_n)$ (si ricordi il limite superiore della costante di Lebesgue calcolata sui punti di Fekete); inoltre, la loro distribuzione asintotica è la stessa dei punti di Fekete nel caso continuo, nel senso che la corrispondente misura di probabilità discreta converge debolmente alla misura di equilibrio pluripotenziale di Ω;

Nei casi pratici le WAMs sono usualmente preferite alle AMs per la loro minore cardinalità . Negli esempi seguenti useremo come WAM sul quadrato i Padua Points (definiti ad esempio in (1)) avente grado n, cardinalità $\frac{(n+1)(n+2)}{2}$ e costante $C(Pad_n) = \mathcal{O}(\frac{2}{\pi}\log^2 n)$.

Estrarre i punti di Fekete da una WAM rimane comunque un problema computazionalmente costoso. Infatti sia $M = \binom{n+d}{d} = dim(\mathbb{P}_n)$ e $N = card(A_n)$, $\phi_1, \ldots, \phi_M$ base ordinata di $\mathbb{P}$, consideriamo la matrice di Vandermonde

$$V(\mathbf{a}; \Phi) = [\phi_j(a_i)], 1 \leqslant i \leqslant N, 1 \leqslant j \leqslant M$$

dove $\mathbf{a}$ è l'array dei punti della mesh. Considerando Φ come vettore colonna, le righe di questa matrice corrisponderanno ai punti della mesh e le colonne agli elementi della base di $\mathbf{P}$, e il problema del calcolo dei punti di Fekete da una WAM si traduce nel selezionare M colonne su N tale che il volume generato da tali colonne sia massimo, cioè che il valore assoluto del determinante della sottomatrice MxM sia massimo. Questo problema non è di facile soluzione, e si

richiedono algoritmi stocastici e euristici.
Sorprendentemente invece si possono trovare buone soluzioni approssimate dette *insiemi estremali discreti*, con semplici procedure di algebra lineare numerica.
La prima procedura, che calcola i punti di Fekete approssimati, massimizza i volumi delle sottomatrici di Vandermonde e può essere implementata tramite la fattorizzazione QR con pivoting sulle colonne della matrice di Vandermonde trasposta. Questa operazione in Matlab è applicata semplicemente usando il comando *backslash* \ per la soluzione di sistemi sovradeterminati.
La seconda procedura calcola i punti discreti di Leja, e si implementa massimizzando il quadrato dei determinanti delle sottomatrici, che equivale alla fattorizzazione standard LU con pivoting sulle righe.
Le procedure sopra descritte sono riassunte nei seguenti pseudo-codici di Matlab (per maggiori dettagli si veda ad esempio (1), (2), (10)).

- *punti approssimati di Fekete* AFP:
  W=$V^t(\mathbf{a};\Phi)$; **b**=ones(1:M);ω=W**b**;ind=find($\omega \neq 0$);$\xi = \mathbf{a}(ind)$

- *punti di Leja discreti* DLP:
  V=V($\mathbf{a};\Phi$);[L,U,σ]=LU(V,*vector*);ind=$\sigma(1,\ldots,M)$;$\xi = \mathbf{a}(ind)$

Nell'algoritmo AFP è possibile prendere nel sistema qualsiasi vettore **b** non nullo, mentre nell'algoritmo DLP usiamo la versione di fattorizzazione LU con pivoting sulle righe che produce un vettore di permutazione sulle righe.
I codici descritti dipendono dalla matrice di Vandermonde di conseguenza dalla scelta della base Φ. Una scelta *sbagliata* della base può produrre dei *cattivi* insiemi di punti, dovuti al possibile malcondizionamento della matrice di Vandermonde. Si può quindi usare preliminarmente una ortogonalizzazione iterata (per maggiori dettagli si può vedere ad esempio (10)):

- algoritmo di ortogonalizzazione iterata:
  V=V($\mathbf{a};\mathbf{q}$);[Q_1,R_1]=qr(V,0);[Q_2,R_2]=qr(Q_1,0);T=inv($R_2 * R_1$)

il quale produce un cambio della base dei polinomi da $\mathbf{q}$ a $\mathbf{p} = T^t\mathbf{q}$, ortonormale rispetto al prodotto interno $\langle f,g\rangle = \sum_{i=1}^{N} f(a_i)\overline{g(a_i)}$ (la fattorizzazione QR usata ha la matrice Q di dimensione NxM e la matrice R triangolare superiore MxM).
La nuova matrice di Vandermonde nella nuova base

$$V(\mathbf{a};\mathbf{p}) = V(\mathbf{a};\mathbf{q})T = Q_2$$

è una matrice numericamente ortogonale unitaria, $\overline{Q_2^t}Q_2 = I$. Normalmente sono sufficienti due iterazioni, a meno che la matrice di partenza non sia eccessivamente malcondizionata (numero di condizionamento superiore al reciproco della precisione di macchina), nel qual caso l'algoritmo fallisce.

Data una WAM sulla quale applicare l'algoritmo, a differenza dei punti di Fekete nel caso continuo, i punti approssimati di Fekete dipendono dalla scelta della base, e i punti di Leja discreti anche dalla scelta dell'ordinamento.
Ad ogni modo entrambe queste famiglie di insiemi estremali discreti hanno lo stesso comportamento asintotico, esattamente quello dei punti di Fekete nel caso continuo: le corrispondenti misure discrete convergono debolmente alla misura di equilibrio pluripotenziale sull'insieme compatto di partenza Ω.

2.3 WAMs su quadrangoli e triangoli

Dopo aver definito la WAM su un quadrato, possiamo utilizzare la terza proprietà per trasformarla in una WAM su un triangolo o un quadrilatero convesso qualsiasi, in virtù delle seguenti due trasformazioni polinomiali:

- trasformazione su un triangolo generico: la mappa quadratica più comunemente utilizzata per *trasformare* il quadrato $[-1,1]$x$[-1,1]$ in triangolo è la trasformazione di Duffy $\mathbf{D}$. Siano $\mathbf{u} = (u_1, u_2)$, $\mathbf{v} = (v_1, v_2)$, $\mathbf{w} = (w_1, w_2)$ i vertici del triangolo.

  $$\mathbf{D}(x,y) = \frac{1}{4}(\mathbf{v}-\mathbf{u})(1+x)(1-y) + \frac{1}{2}(\mathbf{w}-\mathbf{u})(1+y) + \mathbf{u}$$

  ove (x, y) sono le coordinate del quadrato dato. Si vede facilmente che un lato del quadrato degenera in un vertice del triangolo risultante (se ad esempio calcoliamo i Padua points nel quadrato $[-1,1]^2$, il lato $y = 1$ collassa nel vertice $\mathbf{w}$).

- trasformazione su un quadrilatero convesso: si consideri la seguente mappa quadratica $\mathbf{Q}$ che trasforma il quadrato $[-1,1]$x$[-1,1]$ nel quadrilatero di vertici $\mathbf{u} = (u_1, u_2)$, $\mathbf{v} = (v_1, v_2)$, $\mathbf{w} = (w_1, w_2)$, $\mathbf{s} = (s_1, s_2)$

  $$\mathbf{Q}(x,y) = \frac{1}{4}((1-x)(1-y)\mathbf{u}+(1+x)(1-y)\mathbf{v}+(1+x)(1+y)\mathbf{w}+(1-x)(1+y)\mathbf{s})$$

  dove $(x, y) \in [-1,1]^2$. Tale mappa generalizza la trasformazione di Duffy precedentemente introdotta.

La seguente proposizione, infatti, generalizza i Padua points ad un qualsiasi quadrilatero convesso utilizzando le precedenti trasformazioni e le proprietà delle WAM sopra viste (per una dimostrazione formale si veda (7)):

Proposizione

La sequenza delle griglie *oblique* di Chebyshev-Lobatto

$$\mathbf{A}_n = \{\sigma(\xi_j, \xi_k), 0 \leqslant j, k \leqslant n\}, \xi_s = \cos\frac{s\pi}{n}$$

sono una WAM su ogni quadrangolo convesso

$$\mathsf{conv}(\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3, \mathbf{P}_4) = \left\{(x, y) = \sum c_1 \mathbf{P}_i, c_i \geqslant 0, \sum c_i = 1, 1 \leqslant i \leqslant 4\right\}$$

con $C(\mathbf{A}_n) = (\frac{2}{\pi}\log(n+1)+1)^2$ e cardinalità n^2+2n+1) (che si riduce a n^2+n+1 nel caso del triangolo).

A questo punto siamo in grado di ottenere WAMs su qualsiasi quadrangolo convesso e triangolo di cardinalità $\mathcal{O}(2n^2)$ e $C(A_n) = \mathcal{O}(\log^2 n)$.

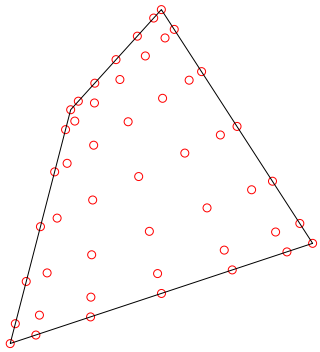

Figura 15: *Esempio di una WAM su un quadrangolo.* 7 x 7 = 49 *punti di Chebyshev-Lobatto per il grado 6.*

2.4 Interpolazione e cubatura su poligoni

Dato quindi un poligono qualsiasi, anche non convesso, possiamo ora costruire una WAM su di esso, suddividendolo in quadrilateri convessi e triangoli utilizzando l'algoritmo sopra esposto e calcolando su ognuno di essi una WAM. Sia quindi il poligono suddiviso in m quadrangoli, sappiamo dalle proprietà sopra menzionate che la WAM A_n trovata avrà cardinalità $\mathcal{O}(2mn^2)$, e $C(A_n) = \max_{i=1,\ldots,m} C(A_{n,i}) = \mathcal{O}(\log^2 n)$ ove le $A_{n,i}$ sono le WAM delle singole suddivisioni.

Osservando la cardinalità risultante da una unione di WAMs, si comprende l'importanza di disporre di una suddivisione minimale dei poligoni. Per poligoni convessi la suddivisione descritta nell'algoritmo *divpolyconvex* risulta essere ottimale. Per poligoni concavi, l'algoritmo di cui abbiamo parlato riesce a suddividere un qualsiasi poligono normale, in particolare ha la proprietà di non dover richiedere alcuna condizione al poligono. La stima dipende dal numero di angoli concavi, ma risulta essere in ogni caso minore della triangolazione, e usualmente la differenza tra i due algoritmi risulta essere considerevole.

Una volta noto un array $\xi = (\xi_1, \ldots, \xi_M)$ corrispondente a un insieme estremale discreto di grado n calcolato da una WAM su un poligono K grazie agli algoritmi AFP e DLP, possiamo interpolare una funzione risolvendo il sistema

$$V(\xi; \mathbf{p})\mathbf{c} = f$$

considerando la possibilità di migliorare il condizionamento della matrice tramite l'ortogonalizzazione sopra descritta.

Inoltre il corrispondente array di pesi ω di cubatura algebrica rispetto ad una misura data μ su K, che sono i pesi di una formula esatta per polinomi di grado non superiore ad n, possono essere calcolati risolvendo il sistema lineare

$$V^t(\xi; \mathbf{p})\omega = \mathbf{m}$$

dove $\mathbf{m}$ sono i momenti della base dei polinomi

$$\mathbf{m} = \int_K \mathbf{p}(x, y)d\mu = T^t \int_K \mathbf{q}(x, y)d\mu$$

Nell'algoritmo AFP per i punti di Fekete approssimati, si può sostituire $\mathbf{m}$ con il vettore non nullo $\mathbf{b}$. Mentre nel caso del secondo algoritmo DLP dei punti discreti di Leja, è sufficente risolvere i sistemi triangolari

$$U^t\mathbf{z} = \mathbf{m}, L^t(:, 1 : M)\omega = \mathbf{z}$$

Riguardo il calcolo dei momenti, ci possiamo riferire per esempio al recente algoritmo citato in (9), basato sulla formula di integrazione di Green e la quadratura di Gauss univariata, che sono in grado di calcolare momenti esatti di una base di polinomi su generici poligoni semplici.
La cubatura numerica su poligoni è generalmente pensata come applicazione della triangolazione di poligoni e cubatura sui triangoli. La presente cubatura può essere vista come nuovo approccio alla (non minimale) cubatura algebrica basata sulla quadrangolazione di poligoni.

2.5 Risultati numerici

Per verificare la *qualità* degli insiemi estremali discreti, abbiamo calcolato numericamente la norma del corrispondente operatore di interpolazione e del funzionale di cubatura, vale a dire la costante di Lebesgue e la somma dei valori assoluti dei pesi di cubatura (si veda (7)). I pesi di cubatura non sono tutti positivi, ma si nota che i negativi sono pochi e relativamente piccoli.
Online è possibile trovare in (6) un codice chiamato POLYGINT che dati i vertici di un poligono in senso antiorario e il grado del polinomio, calcola i punti approssimati di Fekete e i punti discreti di Leja, insieme ai corrispondenti pesi di cubatura, tramite gli algoritmi esposti in queste pagine.
Nella tabella 1 riportiamo le norme dell'operatore di interpolazione e del funzionale di cubatura per vari gradi del polinomio, riferendoci alla Figura 16.
Nellfa Figura 16 mostriamo nuovamente la quadrangolazione tramite l'algoritmo sopra descritto, e gli insiemi estremali discreti calcolati per il grado $n = 15$ (136 punti approssimati di Lebesgue e di Leja discreti). Notiamo che tramite questa triangolazione la WAM risultante è composta da $24n^2$ punti invece che da $35n^2$ risultanti da una triangolazione.
Dalle tabelle possiamo verificare che la costante di Lebesgue è molto minore rispetto alla stima teorica dei punti di Fekete estratti da una WAM, $\Lambda_n \leqslant MC(A_n) = (n+1)(n+2)(\frac{2}{\pi}\log^2(n+1)+1)^2/2$ (proprietà 4 delle WAM e Proposizione).
Vediamo come i punti approssimati di Fekete danno una costante di Lebesgue migliore dei punti di Leja discreti, e una norma leggermente migliore del funzionale di cubatura.
Ad ogni modo entrambi danno buoni risultati sia per l'interpolazione che per la cubatura, come si può vedere dagli errori riportati nelle Tabelle 2 e 3 (per altri esperimenti numerici si vedano ad esempio (8) e (5)).

Tabella 1: Costante di Lebesgue e somma del valore assoluto dei pesi di cubatura e insiemi estremali discreti per il poligono in Figura 16. L'area del poligono ($= \sum \omega_j$) è approssimativamente 6.3e-2.

	points	$n = 3$	$n = 6$	$n = 9$	$n = 12$	$n = 15$	$n = 18$
Λ_n	AFP	3.6	7.3	13.2	18.4	26.8	42.2
	DLP	7.0	10.2	26.0	35.1	44.6	78.7
$\sum \lvert w_j \rvert$	AFP	7.9e-2	6.8e-2	7.0e-2	6.9e-2	7.3e-2	7.0e-2
	DLP	7.0e-2	1.0e-1	1.0e-1	8.2e-2	1.2e-1	8.9e-2

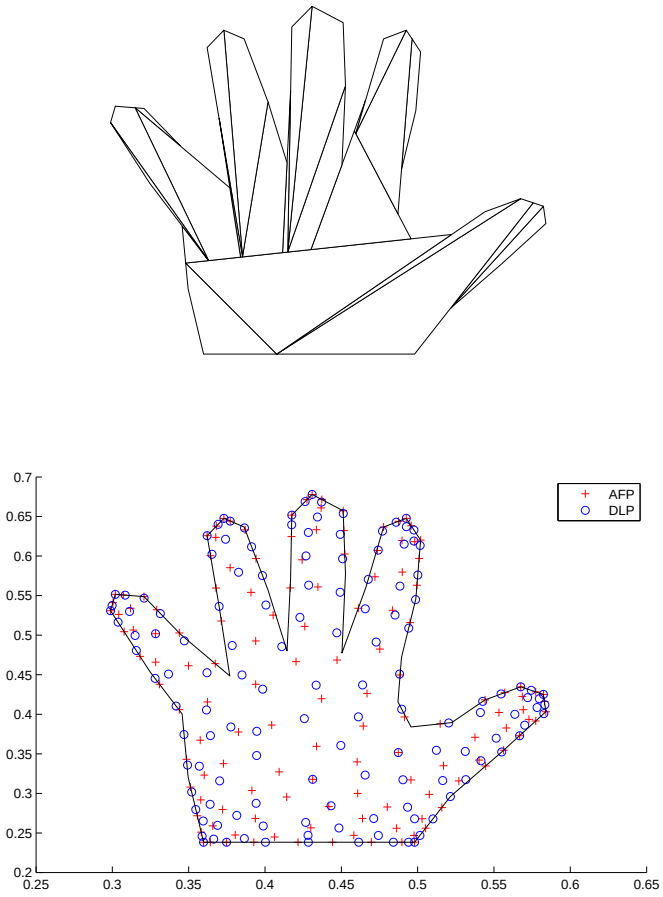

Figura 16: *Quadrangolazione e insiemi estremali discreti di grado 15 di un poligono a forma di mano. 24 quadrangoli per un poligono di 37 lati, e 136 punti approssimati di Lebesgue (+) e discreti di Leja (∘)*

Tabella 2: Interpolazione (errore assoluto) e cubatura (errore relativo) per la funzione $f(x, y) = \cos(x + y)$ nel poligono in Figura 16. Il valore dell'integrale ($d\mu = dx\, dy$) è 4.08555342188148262749791 68e-2.

	points	$n = 3$	$n = 6$	$n = 9$	$n = 12$	$n = 15$	$n = 18$
intp err	AFP	6.4e-5	6.6e-10	5.1e-15	1.3e-15	1.9e-15	1.9e-15
	DLP	8.1e-5	9.1e-10	7.6e-15	1.2e-15	1.6e-15	2.4e-15
cub err	AFP	4.1e-6	4.6e-12	1.0e-15	2.4e-15	2.0e-15	1.0e-15
	DLP	1.3e-7	6.5e-12	6.8e-16	2.7e-16	2.7e-15	2.9e-15

Tabella 3: Come in Tabella 2 utilizzando ora la funzione $f(x, y) = ((x - 0.45)^2 + (y - 0.4)^2)^{3/2}$; il valore dell'integrale è 1.59153824469955942379206 79e-4.

	points	$n = 3$	$n = 6$	$n = 9$	$n = 12$	$n = 15$	$n = 18$
intp err	AFP	1.5e-3	1.2e-4	2.4e-5	1.3e-5	7.0e-6	3.5e-6
	DLP	2.9e-3	9.2e-5	3.9e-5	1.5e-5	9.3e-6	4.5e-6
cub err	AFP	5.4e-2	5.9e-3	3.8e-4	6.9e-5	3.4e-6	1.5e-5
	DLP	4.7e-2	5.4e-3	3.0e-4	1.7e-4	5.9e-5	8.0e-6

3 Esempi grafici

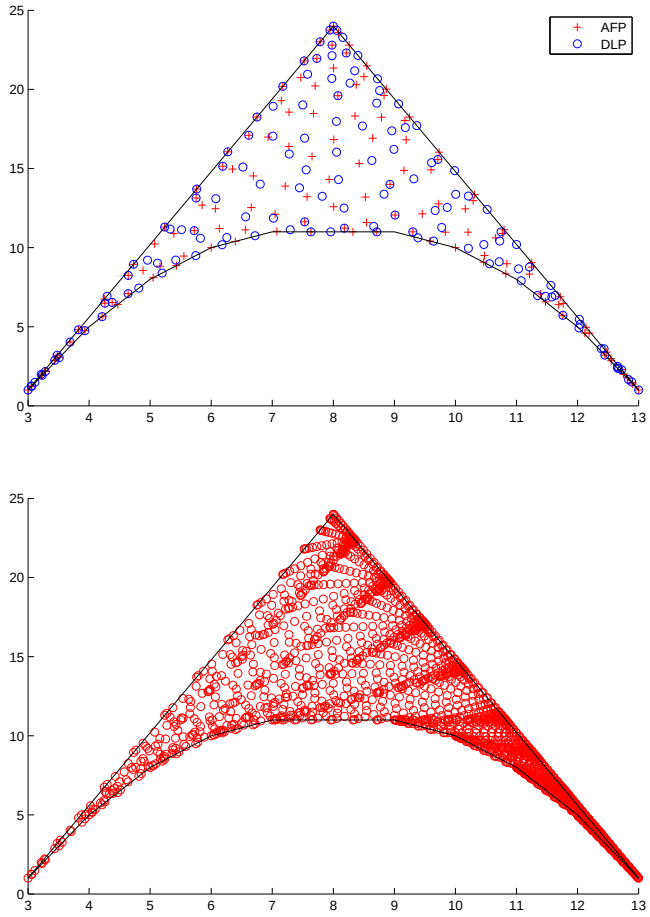

Figura 17: *Calcolo dei punti di Fekete (+) e dei punti di Leja (∘). Il grado utilizzato è 15 e avremo 136 punti di interpolazione, mentre la cardinalità complessiva delle WAM sarà 2169*

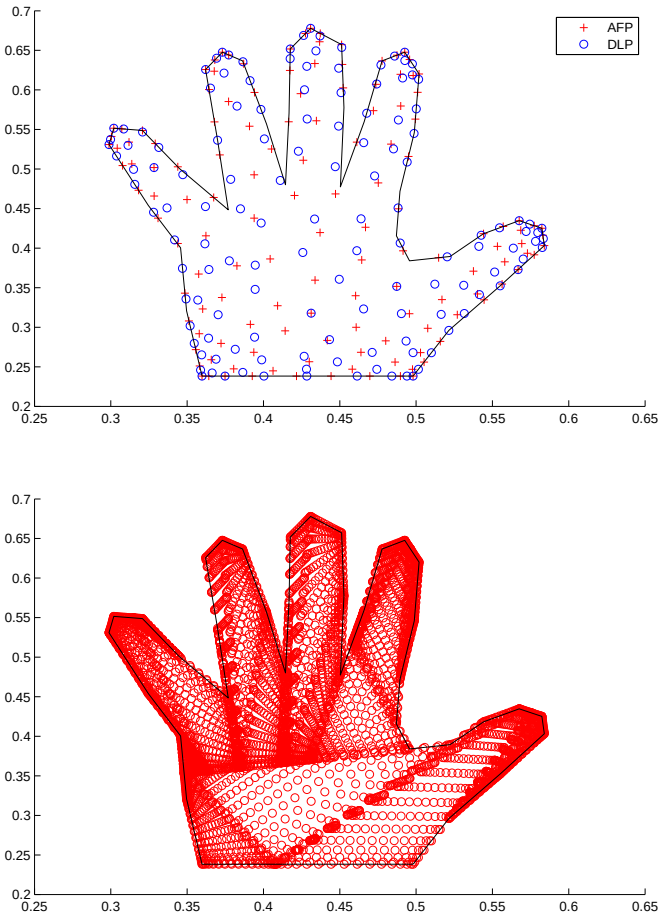

Figura 18: *Figura a forma di mano. Il grado è sempre 15, quindi i punti di Fekete e di Leja (contrassegnati sempre rispettivamente con + e ∘) saranno sempre 136, mentre la cardinalità della WAM risultante è 6312*

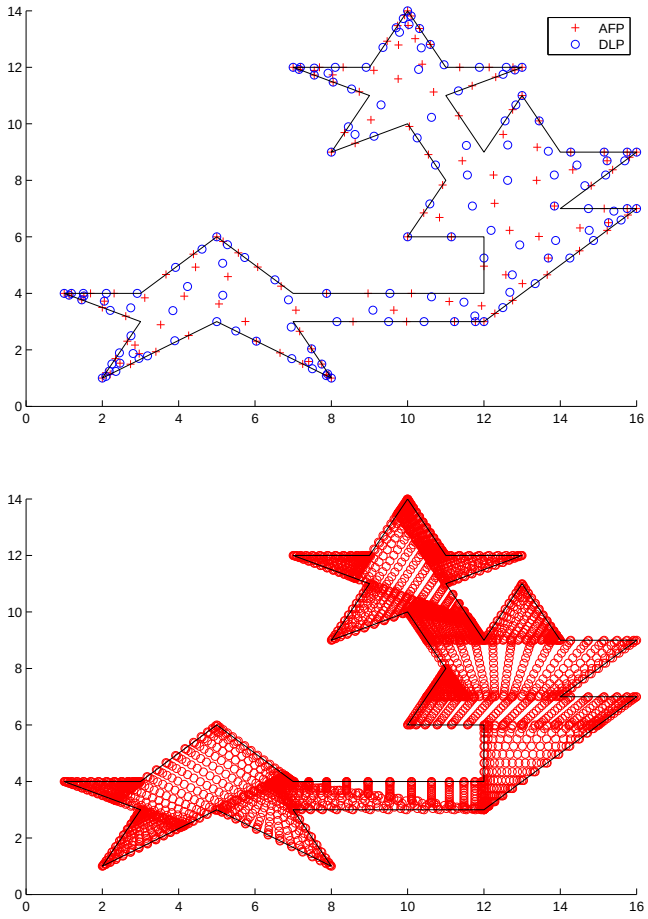

Figura 19: *Utilizziamo ancora il grado 15, e otteniamo di nuovo 136 punti di Fekete e di Leja, con una WAM di cardinalità 4867*

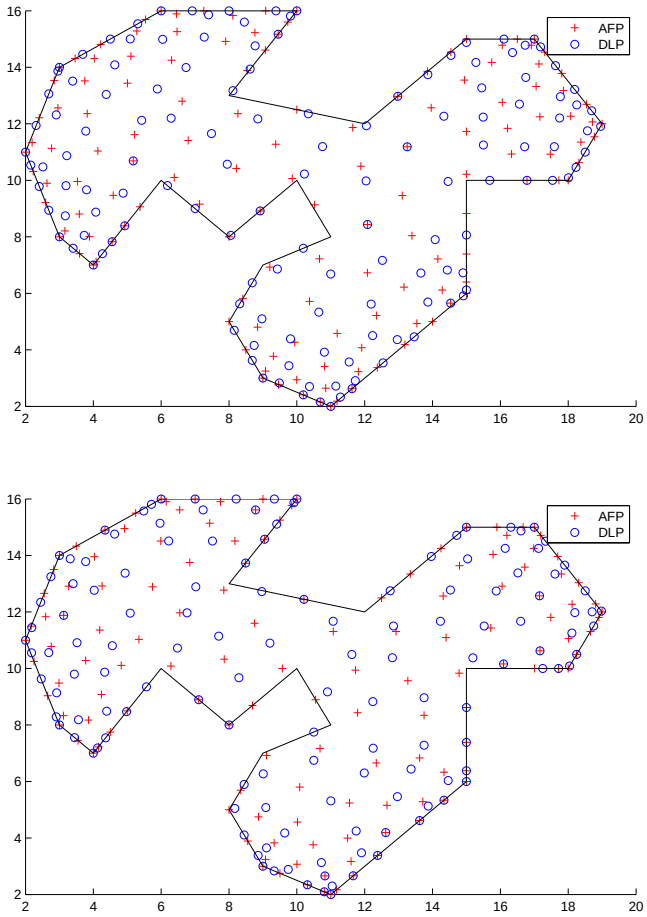

Figura 20: *Anche in queste figure i punti di Fekete e di Leja sono sempre 136, il grado 15. La prima figura è ottenuta sfruttando l'algoritmo qui esposto, mentre la seconda usando una triangolazione.*

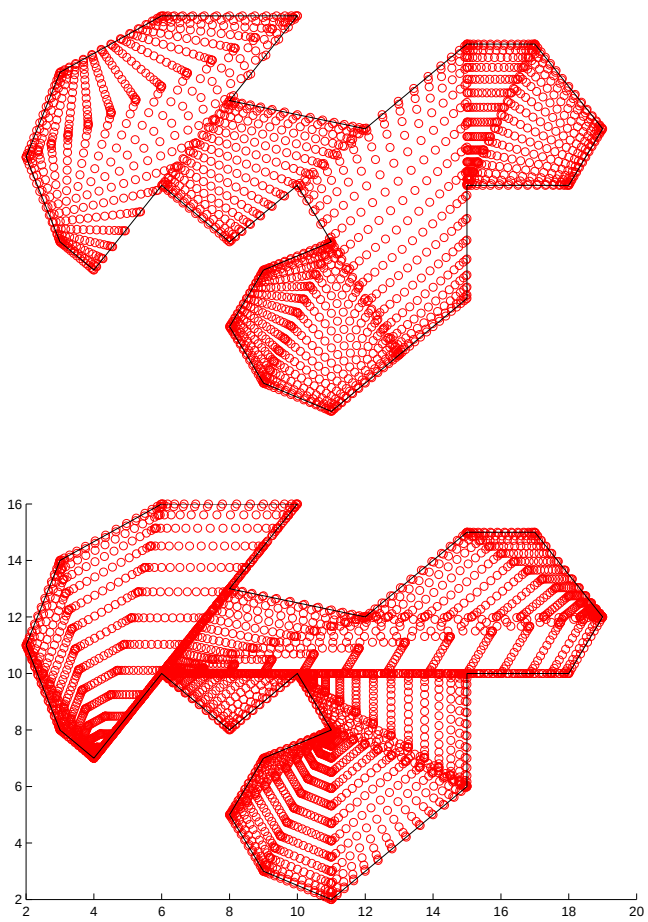

Figura 21: *Qui sono disegnati i punti delle WAMs da cui si sono estratti i punti di Fekete e di Leja sopra disegnati, considerando sempre il grado 15. In questo esempio possiamo apprezzare il quadrangolatore sopra esposto rispetto ad un triangolatore, infatti nel primo caso con la quadrangolazione otteniamo una WAM di 2264 punti, mentre nella figura in basso con la triangolazione risultano essere 4820 punti.*

4 Codice implementato in Matlab

Inserisco di seguito il codice descritto nelle precedenti pagine. Per comodità è stato diviso in varie parti:

- *divconcavitot.m* è la funzione da richiamare per far partire l'algoritmo, che richiamerà tutte le successive. Questa parte dell'algoritmo si limita a gestire le varie matrici temporanee per dare il giusto input alle altre funzioni, e organizza anche l'output. Itera la funzione *divconcavi.m* fino ad avere solo poligoni convessi per poi eseguire su oguno di essi la funzione *divpolyconvex.m*. Infine effettua un controllo di sicurezza per evitare di avere poligoni senza area, ossia 3 o 4 punti allineati. Come ultima cosa, crea un'immagine del poligono diviso, in modo da poter visualizzare graficamente l'output.

- *divconcavi.m* effettua la prima fase dell'algoritmo.

- *divpolyconvex.m* effettua la seconda fase dell'algoritmo.

- *mod1.m* dati i valori b e c essa restituisce il valore $a = b\ mod(c)$ con la modifica che $a = c$ invece di 0 se siamo nel caso $b = c$.

La funzione *divconcavitot.m* si richiama con il seguente comando da utilizzare in Matlab: [A,b]=divconcavitot(c,1).

```
function [A,b]=divconcavitot(c,figura)

% SCOPO:
%
% dividere un poligono qualsiasi (anche concavo), normale (senza lati che
% si autointersecano) in quadrilateri e triangoli (cercando di minimizzare
% il numero di questi ultimi)
%
%
% NOTE SUL PROCEDIMENTO
%
% per ogni angolo concavo, si prolunga il primo dei suoi
% lati fino alla prima intersezione con la frontiera del
% poligono (prolungando verso l'interno del poligono stesso)
% l'angolo concavo risultera' quindi diviso in un angolo piatto
% e un angolo convesso
% finito questo procedimento, ogni sottopoligono convesso viene diviso
% in quadrilateri e al massimo un triangolo
%
%
% STIMA SULLA SUDDIVISIONE
%
% dati n,k numero vertici e numero angoli concavi del poligono dato,
% la stima superiore dei quadrilateri risultanti e':
%
% ceil(n/2)+floor((k+1)/2)-1
```

```
%
%
% INPUT:
%
% c: vertici del poligono, matrice nx2 contenente ascisse e ordinate del
% poligono [x(i),y(i)] i=1,...n
% il poligono deve essere scritto in senso ANTIORARIO
% il primo vertice NON deve essere ripetuto alla fine del vettore
%
% figura: variabile binomiale 1: disegna il poligono suddiviso
%                             0: non disegna il poligono
%
%
% OUTPUT:
%
% A: matrice di dimensione 4x2h
%    contiene le coordinate dei quadrati risultanti (h=numero quadrati)
%    A=[x(1),y(1),x(2),y(2),.....,x(h),y(h)]
%    (x e y sono vettori colonna)
%    nel caso un quadrato si riduca ad essere un triangolo, le coordinate
%    corrispondenti della matrice A avranno 0 0 nella quarta riga
%
% b: vettore riga di dimensione h (numero di quadrilateri risultanti)
%    e' formato da una serie di 4 e 3, i numeri 3 indicano che le
%    coordinate corrispondenti nella matrice A si riferiscono ad un
%    triangolo
%
%    la colonna j in b si riferisce alle colonne 2j-1 e 2j nella matrice A
%
%
% FUNZIONI USATE NON PRESENTI IN MATLAB
%
% [A,b]=divconcavi(c)
% [A,b]=divpolyconvex(c)
% [a]=mod_1(b,c)
%
% le funzioni citate si trovano di seguito in questo stesso file


tic

A_nc=c;
b_nc(1)=size(c,1);
k_A_1=1; k_b_1=1;
t=1;
while t==1                % si itera finche' si hanno poligoni concavi
    clear c_temp;
    for k=1:1:2
        for h=1:1:b_nc(1)
            c_temp(h,k)=A_nc(h,k);    % si crea un vettore da mettere in input in divconcavi
        end
    end
    clear A_temp; clear b_temp;
    [A_temp,b_temp]=divconcavi(c_temp);   % si divide un angolo concavo (vedi divconcavi)
    if size(A_temp,2)==2
        for h=1:1:b_temp
            A_1(h,k_A_1)=A_temp(h,1);     % riorganizzazione dell'output
            A_1(h,k_A_1+1)=A_temp(h,2);   % A_1 contiene i sottopoligoni convessi
        end
        k_A_1=k_A_1+2;
        b_1(k_b_1)=b_temp;
        k_b_1=k_b_1+1;
```

```
    end
    clear s_A_nc;
    clear s_A_temp;
    s_A_nc=size(A_nc,2);
    s_A_temp=size(A_temp,2);
    if (s_A_nc==2 && s_A_temp==2)         % in questo caso ci si rende conto che non si hanno
        t=0;                              % piu' angoli concavi da dividere - il processo termina
    end
    if (s_A_nc==2 && s_A_temp~=2)
        clear A_nc; clear b_nc;
        A_nc=A_temp;
        b_nc=b_temp;
    end
    if (s_A_nc~=2 && s_A_temp==2)
        clear A_nc_temp; clear b_nc_temp; clear k_b_nc_temp;
        A_nc_temp=A_nc;
        b_nc_temp=b_nc;
        clear A_nc; clear b_nc;
        k_b_nc_temp=1;
        for h=3:2:size(A_nc_temp,2)-1
            k_b_nc_temp=k_b_nc_temp+1;
            b_nc(k_b_nc_temp-1)=b_nc_temp(k_b_nc_temp);
            for k=1:1:b_nc_temp(k_b_nc_temp)
                A_nc(k,h-2)=A_nc_temp(k,h);
                A_nc(k,h-1)=A_nc_temp(k,h+1);
            end
        end
    end
    if (s_A_nc~=2 && s_A_temp~=2)
        clear A_nc_temp; clear b_nc_temp; clear k_b_nc_temp;
        A_nc_temp=A_nc;
        b_nc_temp=b_nc;
        clear A_nc; clear b_nc;
        k_b_nc_temp=1;
        for h=3:2:size(A_nc_temp,2)-1
            k_b_nc_temp=k_b_nc_temp+1;
            b_nc(k_b_nc_temp-1)=b_nc_temp(k_b_nc_temp);
            for k=1:1:b_nc_temp(k_b_nc_temp)
                A_nc(k,h-2)=A_nc_temp(k,h);
                A_nc(k,h-1)=A_nc_temp(k,h+1);
            end
        end
        f_A=size(A_nc_temp,2)-2;
        f_b=length(b_nc_temp)-1;
        for k=1:1:max(b_temp(1),b_temp(2))
            for h=1:1:4
                A_nc(k,h+f_A)=A_temp(k,h);
            end
        end
        b_nc(f_b+1)=b_temp(1);
        b_nc(f_b+2)=b_temp(2);
    end
end

k_A_1=1; k_A=0; k_b=0;
for k=1:1:length(b_1)
    clear c_convex;
    for l=1:1:b_1(k)
        c_convex(l,1)=A_1(l,k_A_1);    % si crea l'input per divpolyconvex
        c_convex(l,2)=A_1(l,k_A_1+1);  % la matrice A_1 contiene solo poligoni convessi
    end
    k_A_1=k_A_1+2;
```

```
    clear A_convex; clear b_convex;
    [A_convex,b_convex]=divpolyconvex(c_convex); % si richiama divpolyconvex la quale divide ogni
    for m=1:1:length(b_convex)                   % poligono convesso in quadrati convessi
                                                 % (e al max un triangolo)
        b(k_b+m)=b_convex(m);       % si crea il vettore b
    end
    k_b=k_b+length(b_convex);

    for m=1:1:size(A_convex,2)
        for l=1:1:size(A_convex,1)
            A(l,k_A+m)=A_convex(l,m);        % la matrice A sara' l'output finale, contenente
        end                                  % solo quadrati convessi (alcuni ridotti a triangoli)
    end
    k_A=k_A+size(A_convex,2);
end
if max(b)==3
    A(4,1)=0;
end
s_A=size(A,2);        % eliminazione poligoni "senza area"
clear k_b;
k_b=0;
for k=1:2:s_A-1
    k_b=k_b+1;
    clear v_1; clear v_2; clear v_3; clear v_4;
    v_1(1)=A(1,k); v_1(2)=A(1,k+1); v_1(3)=0;
    v_2(1)=A(2,k); v_2(2)=A(2,k+1); v_2(3)=0;
    v_3(1)=A(3,k); v_3(2)=A(3,k+1); v_3(3)=0;
    v_4(1)=A(4,k); v_4(2)=A(4,k+1); v_4(3)=0;
    s_b=b(k_b);
    if s_b==3
        if norm(cross(v_3-v_2,v_2-v_1))==0
            b(k_b)=0;
        end
    end
    if s_b==4
        pv_1=norm(cross(v_1-v_4,v_2-v_1));
        pv_2=norm(cross(v_2-v_1,v_3-v_2));
        pv_3=norm(cross(v_4-v_3,v_3-v_2));
        pv_4=norm(cross(v_4-v_3,v_1-v_4));
        if pv_1==0
            b(k_b)=3;
            A(1,k)=A(2,k); A(1,k+1)=A(2,k+1);
            A(2,k)=A(3,k); A(2,k+1)=A(3,k+1);
            A(3,k)=A(4,k); A(3,k+1)=A(4,k+1);
            A(4,k)=0; A(4,k+1)=0;
        end
        if pv_2==0
            b(k_b)=3;
            A(2,k)=A(3,k); A(2,k+1)=A(3,k+1);
            A(3,k)=A(4,k); A(3,k+1)=A(4,k+1);
            A(4,k)=0; A(4,k+1)=0;
        end
        if pv_3==0
            b(k_b)=3;
            A(3,k)=A(4,k); A(3,k+1)=A(4,k+1);
            A(4,k)=0; A(4,k+1)=0;
        end
        if pv_4==0
            b(k_b)=3;
            A(4,k)=0; A(4,k+1)=0;
        end
        if pv_1+pv_2==0
```

```
            b(k_b)=0;
        end
    end
end
A_fin=A; s_A_fin=size(A_fin,2);
b_fin=b;
clear A; clear b;
k_b=1;
k_A=1;
k_b_fin=0;
for k=1:2:s_A_fin-1
    k_b_fin=k_b_fin+1;
    if b_fin(k_b_fin)~=0
        A(:,k_A)=A_fin(:,k);
        A(:,k_A+1)=A_fin(:,k+1);
        b(k_b)=b_fin(k_b_fin);
        k_A=k_A+2;
        k_b=k_b+1;
    end
end

if figura==1             % disegno del poligono suddiviso
k_b=1;
figure(1);
hold on;
for k=1:2:(size(A,2)-1)
    clear x; clear y;
    for h=1:1:3
        x(h)=A(h,k);
        y(h)=A(h,k+1);
    end
    if b(k_b)==4
        x(4)=A(4,k);
        y(4)=A(4,k+1);
    end
    fill(x,y,'w');
    k_b=k_b+1;
end
hold off;
end

toc



function [A,b]=divconcavi(c)

% SCOPO
%
% dividere un poligono concavo in due seguendo questo procedimento:
%
% dopo aver verificato la presenza di un angolo concavo, si prolunga il
% primo dei suoi lati fino alla prima intersezione con la frontiera del
% poligono (prolungando verso l'interno del poligono stesso)...l'angolo
% concavo risultera' quindi diviso in un angolo piatto e un angolo convesso
%
%
% INPUT
%
% c: vettore nx2 contenente le coordinate del poligono
% il poligono deve essere scritto in senso ANTIORARIO
% il primo vertice NON deve essere ripetuto alla fine del vettore
```

```
%
%
%
% OUTPUT
%
% A: dati i due poligoni con numeri di vertici n, m, la matrice avra'
%    dimensione max(n,m)x4, contenente le coordinate dei vertici dei due
%    poligoni risultanti [x(1),y(1),x(2),y(2)]
%    i posti "vuoti" della matrice, al cui posto verra' scritto 0,
%    saranno indicati dal vettore b
%
% b: indica i numeri di vertici dei due poligoni risultanti
%
%
% tutte le operazioni usate in questa funzione sono presenti in Matlab

n=size(c,1);
m=0;
k=1;
while (m==0 && k~=n+1)            % ricerca dell'angolo concavo
    v_1(1)=c(mod_1(k-1,n),1);
    v_1(2)=c(mod_1(k-1,n),2);
    v_1(3)=0;
    v_2(1)=c(mod_1(k,n),1);
    v_2(2)=c(mod_1(k,n),2);
    v_2(3)=0;
    v_3(1)=c(mod_1(k+1,n),1);
    v_3(2)=c(mod_1(k+1,n),2);
    v_3(3)=0;
    v=cross(v_2-v_1,v_3-v_2);
    v_s=-sign(v(3));
    m=floor((v_s+1)/2);
    k=k+1;
end

if m==0
    A=c;                % caso in cui il poligono in input non contiene
    b=size(c,1);        % angoli concavi (gia' convesso)
end
if m==1
    k=k-1;   % angolo concavo
    v_1(1)=c(mod_1(k-1,n),1);
    v_1(2)=c(mod_1(k-1,n),2);
    v_2(1)=c(mod_1(k,n),1);
    v_2(2)=c(mod_1(k,n),2);
    z=(v_2(2)-v_1(2))/(v_2(1)-v_1(1));   % coefficiente angolare del primo
    if z==-Inf                           % lato dell'angolo concavo
        z=Inf;
    end
    s=k+1;
    c_2=[0,0,0,realmax,0,0];            % in c_2 registreremo tutte le intersezioni con la frontiera
    while mod_1(s,n)~=mod_1(k,n)        % del poligono e sceglieremo di volta in volta l'intersezione
         v_1_bis(1)=c(mod_1(s,n),1);    % "piu' vicina" in modo da ottenere alla fine
         v_1_bis(2)=c(mod_1(s,n),2);    % l'intersezione "ovvia"
         v_2_bis(1)=c(mod_1(s+1,n),1);
         v_2_bis(2)=c(mod_1(s+1,n),2);
         controllo=0;
         z_1=(v_2_bis(2)-v_1_bis(2))/(v_2_bis(1)-v_1_bis(1));   % coefficiente angolare dei
            if z_1==-Inf                                        % lati del poligono
                z_1=Inf;
            end
            if z_1==z
```

```
            if v_1_bis(2)-v_2(2)==z*(v_1_bis(1)-v_2(1))
                t=(v_1_bis(1)-v_1(1))/(v_2(1)-v_1(1));
                if t>1
                    if t<c_2(4)                % primo caso
                        c_2(3)=s;              % il prolungamento del lato dell'angolo concavo
                        c_2(4)=t;              % interseca la frontiera in esattamente un vertice
                        c_2(5)=1;              % del poligono
                        c_2(6)=1;
                    end
                end
            end
        end
      if z_1~=z
      if z==Inf
          if (v_1_bis(1)==v_2(1) || (v_1_bis(1)-v_2(1))*(v_2_bis(1)-v_2(1))<=0)
             controllo=1;      % la variabile "controllo" indica se e' effettivamente possibile
          end                  % che il lato in questione intersechi il prolungamento
      end                      % dell'angolo concavo
      if z~=Inf
          if ((v_1_bis(2)-v_2(2)-(z*(v_1_bis(1)-v_2(1))))*...
             (v_2_bis(2)-v_2(2)-(z*(v_2_bis(1)-v_2(1)))))<=0
              controllo=1;
          end
      end
      if controllo==1
            W=[v_2_bis(1)-v_1_bis(1), v_1(1)-v_2(1); v_2_bis(2)-v_1_bis(2), v_1(2)-v_2(2)];
            V=[v_1(1)-v_1_bis(1); v_1(2)-v_1_bis(2)];
            X=W\V;                     % secondo caso
            tol=1e-10;                 % il prolungamento interseca un lato della frontiera
            if X(1)>tol && X(1)<1      % se e' possibile che ci sia intersezione, la
                if X(2)>1              % calcoliamo con un sistema lineare di matrice 2x2
                    if X(2)<c_2(4)
                        c_2(1)=v_1_bis(1)+X(1)*(v_2_bis(1)-v_1_bis(1));
                        c_2(2)=v_1_bis(2)+X(1)*(v_2_bis(2)-v_1_bis(2));
                        c_2(3)=s;
                        c_2(4)=X(2);
                        c_2(5)=0;
                        c_2(6)=0;
                    end
                end
            end
            if X(1)<=tol && X(1)>=-tol   % terzo caso (primo caso bis)
                if X(2)>1                % consideriamo questo caso per eventuali problemi di
                    if X(2)<c_2(4)       % approssimazione in Matlab. E' molto probabile che il
                        c_2(3)=s;        % prolungamento dell'angolo concavo tocchi un vertice
                        c_2(4)=X(2);     % del poligono, ma qualche approssimazione ci ha impedito
                        c_2(5)=1;        % di essere nel primo caso sopra visto
                        c_2(6)=0;
                    end
                end
            end
      end
      end
         s=s+1;     % contatore: il processo termina una volta effettuato per ogni vertice
    end

k_1=k;
t=1;
b(1)=0;
while mod_1(k_1,n)~=mod_1(c_2(3),n)    % creazione dei due poligoni risultanti dalla suddivisione
    A(t,1)=c(mod_1(k_1,n),1);
    A(t,2)=c(mod_1(k_1,n),2);
```

```
    b(1)=b(1)+1;
    t=t+1;
    k_1=k_1+1;
end
if c_2(5)==0
    A(t,1)=c(mod_1(c_2(3),n),1);
    A(t,2)=c(mod_1(c_2(3),n),2);
    b(1)=b(1)+1;
    A(t+1,1)=c_2(1);
    A(t+1,2)=c_2(2);
    b(1)=b(1)+1;
end
if c_2(5)==1
    if z==Inf
        if c(mod_1(k,n),1)~=c(mod_1(c_2(3)-1,n),1)
            A(t,1)=c(mod_1(c_2(3),n),1);
            A(t,2)=c(mod_1(c_2(3),n),2);
            b(1)=b(1)+1;
        end
    end
    if z~=Inf
        if c(mod_1(c_2(3),n),2)~=((c(mod_1(c_2(3)-1,n),2)-...
           c(mod_1(k,n),2))/(c(mod_1(c_2(3)-1,n),1)-c(mod_1(k,n),1)))*...
           (c(mod_1(c_2(3),n),1)-c(mod_1(k,n),1))+c(mod_1(k,n),2)
            A(t,1)=c(mod_1(c_2(3),n),1);
            A(t,2)=c(mod_1(c_2(3),n),2);
            b(1)=b(1)+1;
        end
    end
end

s_1=c_2(3);
t=1;
b(2)=0;
if c_2(5)==0
    A(t,3)=c_2(1);
    A(t,4)=c_2(2);
    b(2)=b(2)+1;
    t=t+1;
    s_1=s_1+1;
end
if c_2(5)==1
    if z==Inf
        if c(mod_1(k,n),1)==c(mod_1(s_1,n),1) && c(mod_1(k,n),1)==c(mod_1(s_1+1,n),1)
            s_1=s_1+1;
        end
    end
    if z~=Inf
        if c(mod_1(s_1,n),2)==((c(mod_1(s_1+1,n),2)-c(mod_1(k,n),2))/...
           (c(mod_1(s_1+1,n),1)-c(mod_1(k,n),1)))*(c(mod_1(s_1,n),1)...
           -c(mod_1(k,n),1))+c(mod_1(k,n),2)
            s_1=s_1+1;
        end
    end
end
while mod_1(s_1,n)~=mod_1(k,n)
    A(t,3)=c(mod_1(s_1,n),1);
    A(t,4)=c(mod_1(s_1,n),2);
    b(2)=b(2)+1;
    t=t+1;
    s_1=s_1+1;
end
```

```
end


function [A,b]=divpolyconvex(c)

% SCOPO
%
% dividere un poligono convesso in quadrilateri seguendo questo
% procedimento:
%
% partendo dal primo vertice, e ruotando in senso antiorario, si
% raggruppano i vertici a gruppi di 4...ad ogni iterazione il poligono
% restante avra' n-2 vertici, se n era il numero iniziale di vertici
%
% nel caso il poligono abbia un numero dispari di vertici, l'ultimo
% quadrilatero risultera' in effetti un triangolo
%
%
% INPUT
%
% c: vettore nx2 contenente le coordinate del poligono
% il poligono deve essere scritto in senso ANTIORARIO
% il primo vertice NON deve essere ripetuto alla fine del vettore
%
%
% OUTPUT
%
% A: matrice contenente le coordinate dei quadrilateri risultanti...nel
%    caso l'ultimo fosse in triangolo, le ultime 2 coordinate saranno 0 0
% b: vettore riga formato da 4 e 3 (l'eventuale 3 sara' in ultima
%    posizione)...il 3 indica che l'ultimo quadrilatero e' in effetti un
%    triangolo, e avvisa quindi di non considerare le ultime coordinate 0 0
%
%
% tutte le operazioni usate in questa funzione sono presenti in Matlab



if size(c,2) ~= 2  % controllo sul vettore c
print('error in divpolyconvex - controllo sul vettore in input');
end

k=1; c_1=c; c_2=c_1; k_b=1;

while size(c_1,1) >= 4   % algoritmo di divisione
clear c_2;
for l=1:1:4
A(l,k)=c_1(l,1);
A(l,k+1)=c_1(l,2);
end
b(k_b)=4;
c_2(1,1)=c_1(1,1);      % eliminazione dei due vertici "esterni"
c_2(1,2)=c_1(1,2);      % al poligono restante
for m=4:1:size(c_1,1)
c_2(m-2,1)=c_1(m,1);
c_2(m-2,2)=c_1(m,2);
end
c_1=c_2;
k=k+2;
k_b=k_b+1;
end
```

```
if size(c_1,1) == 3   % triangolo di scarto
b(k_b)=3;
for t=1:1:3
A(t,k)=c_1(t,1);
A(t,k+1)=c_1(t,2);
end
end


function [a]=mod_1(b,c)

a=mod(b,c);      % piccola modifica della funzione "modulo"

if a==0
    a=c;
end
```

Riferimenti bibliografici

[1] L. Bos, J.P.Calvi, N.Levenberg, A. Sommariva, and M. Vianello. Geometric weakly admissible meshes, discrete least squares approximation and approximate fekete points. *to appear*.

[2] L. Bos, S. De Marchi, A. Sommariva, and M. Vianello. Computing multivariate fekete and leja points by numerical linear algebra. *SIAM J. Numer. Anal. (to appear)*, 2010.

[3] L. Bos, S. De Marchi, A. Sommariva, and M. Vianello. Weakly admissible meshes and discrete extremal sets. *Numer. Math. Theory Methods Appl. (to appear)*, 2010.

[4] J.P.Calvi and N. Levenberg. Uniform approximation by discrete least squares polynomials,. *J. Approx. Theory*, **152**:82–100, 2008.

[5] L.Bos, S.De Marchi, A. Sommariva, and M. Vianello. On multivariate newton interpolation at discrete leja points. 2011.

[6] M.Gentile, A. Sommariva, and M. Vianello. Polygint: a matlab code for interpolation and cubature at discrete extremal sets of polygons. *online: www.math.unipd.it/ marcov/CAAsoft.html*.

[7] M.Gentile, A. Sommariva, and M. Vianello. Polynomial interpolation and cubature over polygons. *to appear*, 2010.

[8] A. Sommariva and M. Vianello. Approximate fekete points for weighted polynomial interpolation. 2000.

[9] A. Sommariva and M. Vianello. Product gauss cubature over polygons based on green's integration formula. *BIT Numerical Mathematics*, **47**:441–453, 2007.

[10] A. Sommariva and M. Vianello. Computing approximate fekete points by qr factorizations of vandermonde matrices. *Comput. Math. Appl.*, **57**:1324–1336, 2009.