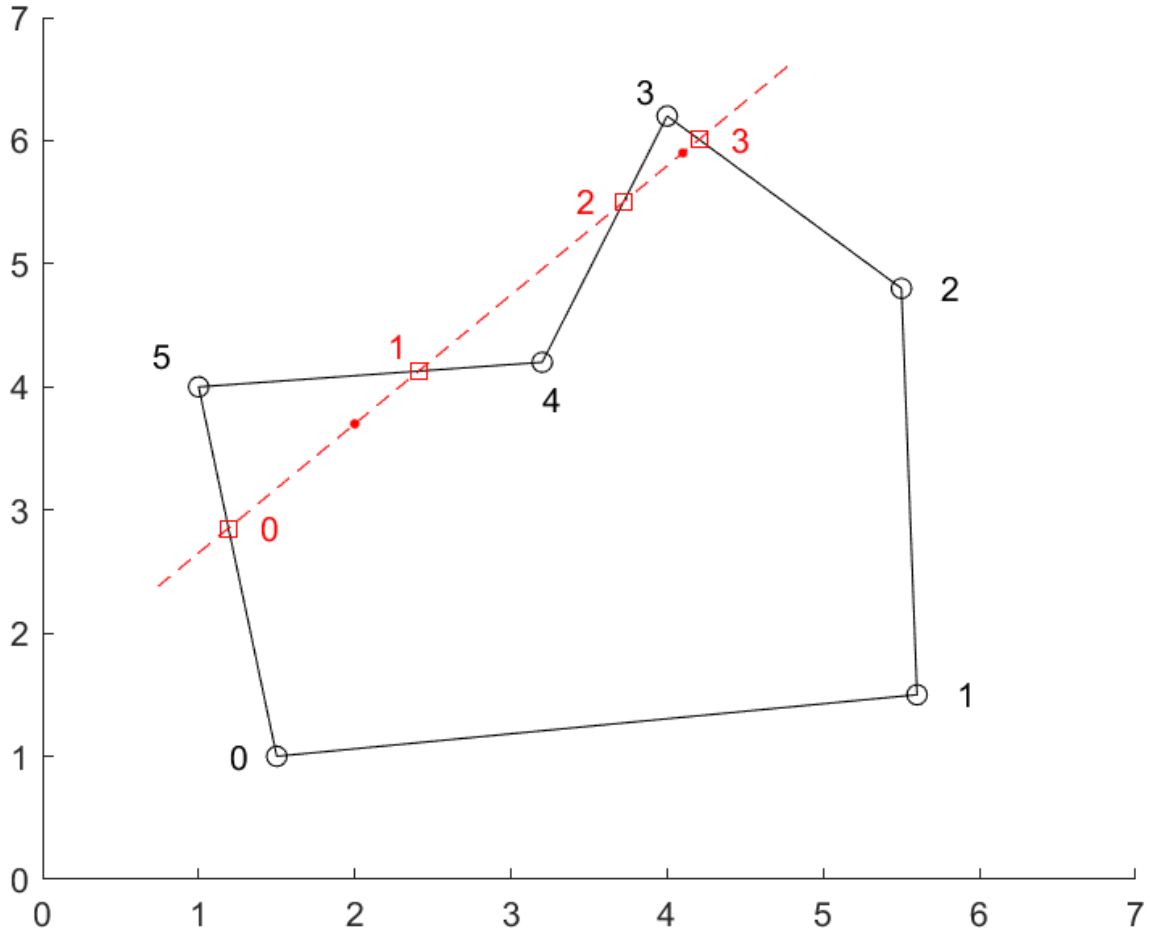
Introduzione

La versione base dell’ algoritmo si occupa di poligoni tagliati da un segmento che non passa attraverso i vertici.

Sotto questa ipotesi i punti di intersezione sono in numero pari e, se ordinati secondo ascissa curvilinea, si alternano tra entrante (con indice pari) ed uscente (con indice dispari) dal poligono, in modo tale che formano coppie di segmenti interni.



Strutture dati di supporto

Ai fini della logica dell'algoritmo sono stati utilizzati tre vettori:

- vector<bool> found

  Indica se il vertice del poligono è stato esplorato.

- vector<int> segToInt

  Gli indici del vettore coincidono con gli indici dei segmenti ed i suoi valori possono essere -1 se non c'è intersezione oppure l' id del punto di intersezione (ordinati tramite ascissa curvilinea).

- vector<int> intToSeg

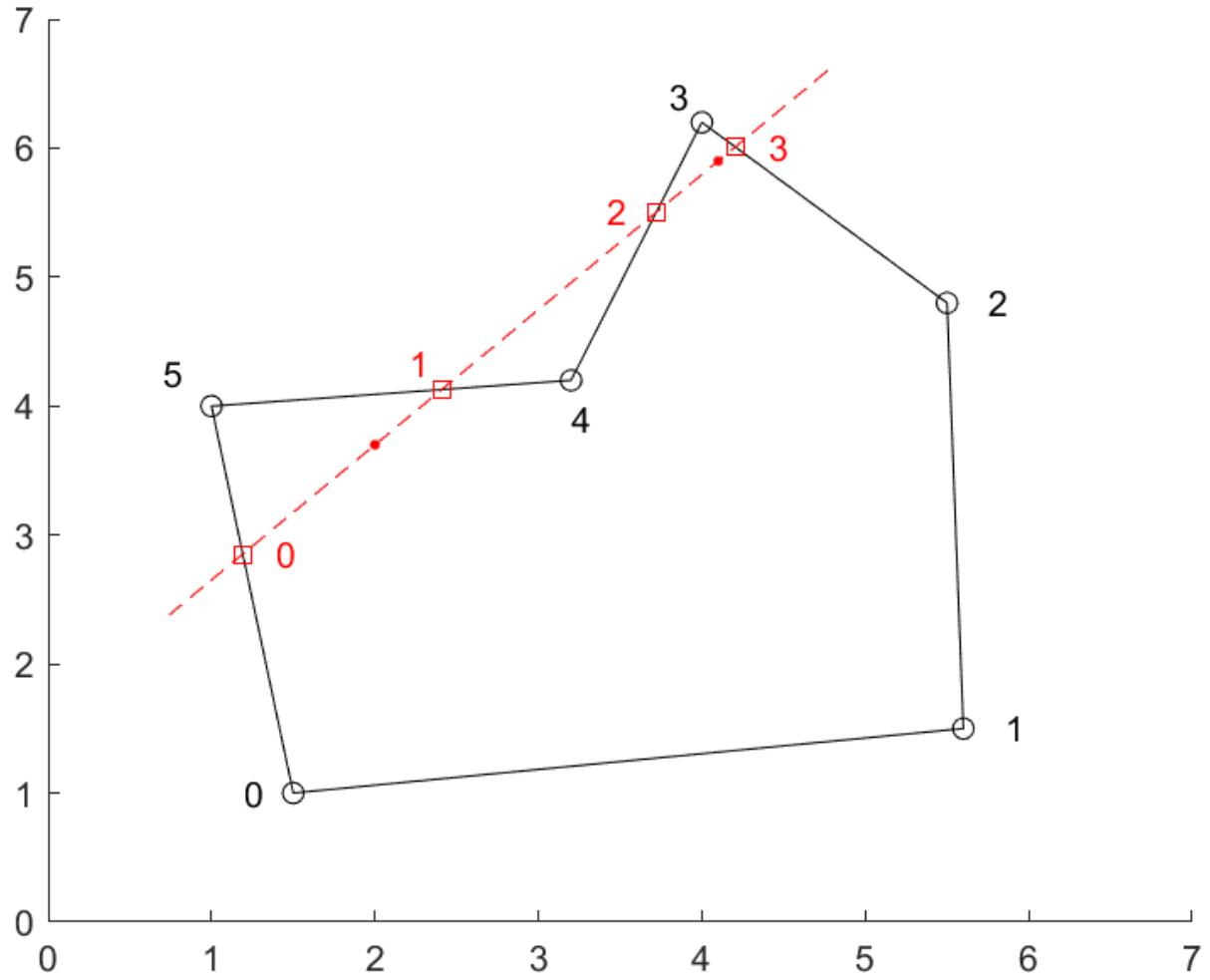
  Gli indici coincidono con l'ordinamento dei punti di intersezione ed i valori mostrano in quale segmento del poligono incide il punto.

	segToInt
0	-1
1	-1
2	3
3	2
4	1
5	0

	intToSeg
0	5
1	4
2	3
3	2

	found
0	0
1	0
2	0
3	0
4	0
5	0

segToInt e intToSeg sono vettori noti, ricavati durante il calcolo dei punti di intersezione. Inoltre inizialmente il vettore found è settato completamente a false.

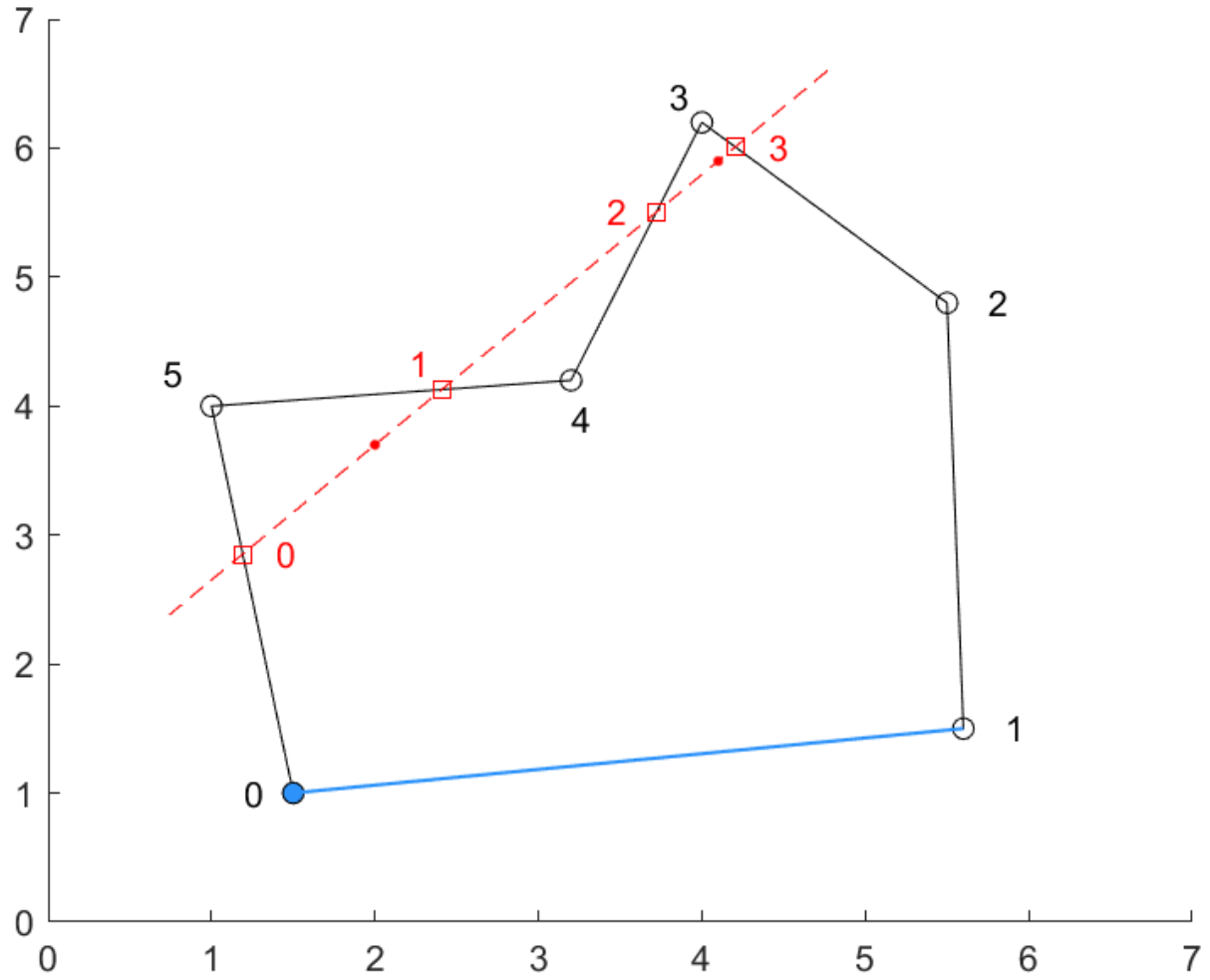


	segToInt
0	-1
1	-1
2	3
3	2
4	1
5	0

	intToSeg
0	5
1	4
2	3
3	2

	found
0	1
1	0
2	0
3	0
4	0
5	0

Il primo punto ad essere percorso è lo 0-esimo ed il rispettivo booleano nel vettore found viene impostato true.

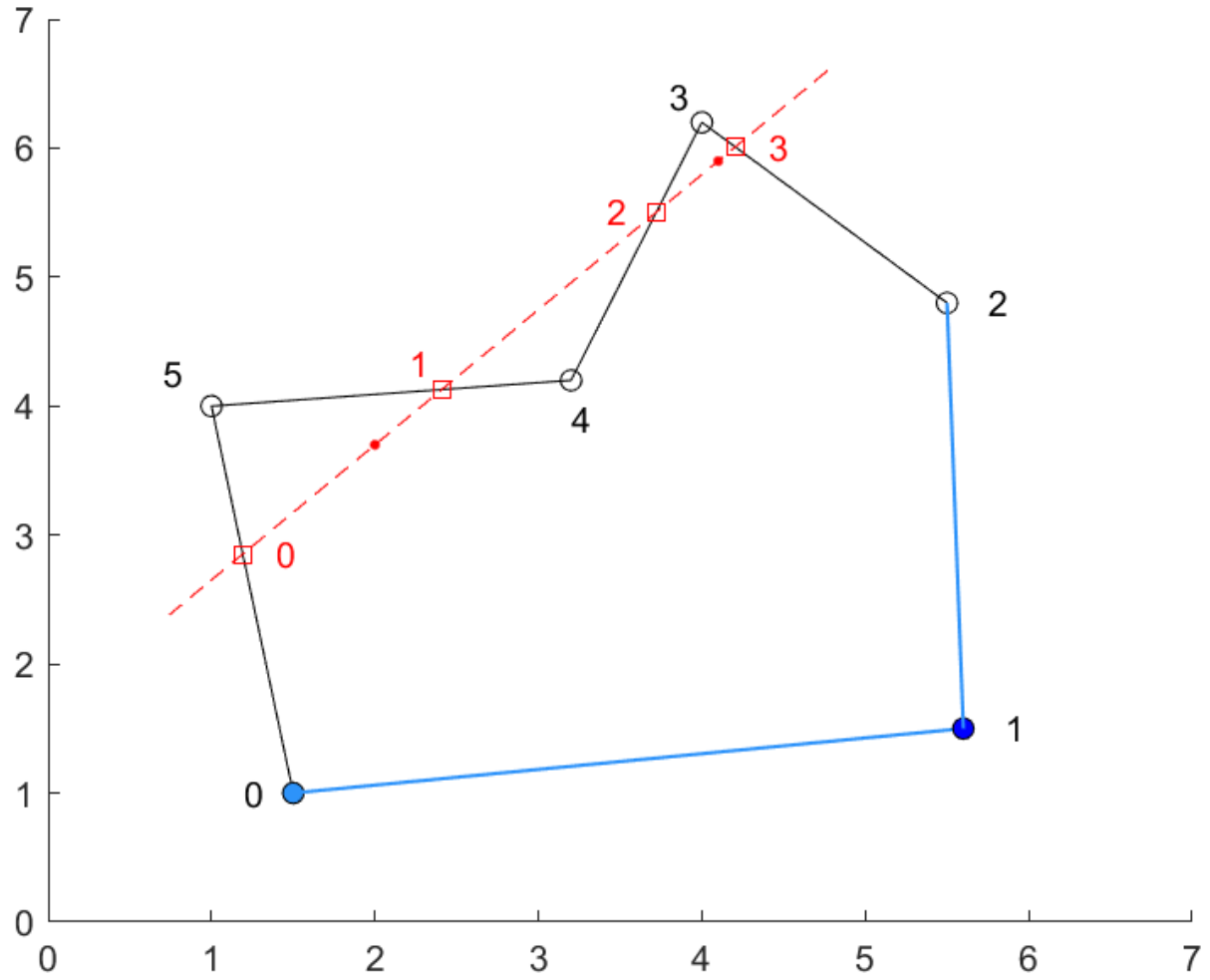


	segToInt
0	-1
1	-1
2	3
3	2
4	1
5	0

	intToSeg
0	5
1	4
2	3
3	2

	found
0	1
1	1
2	0
3	0
4	0
5	0

L’algoritmo passa per il punto 1 ed in quanto non c’è intersezione nel segmento 1, segnalato da segToInt[1] = -1, l’id del punto successivo diventa 2.

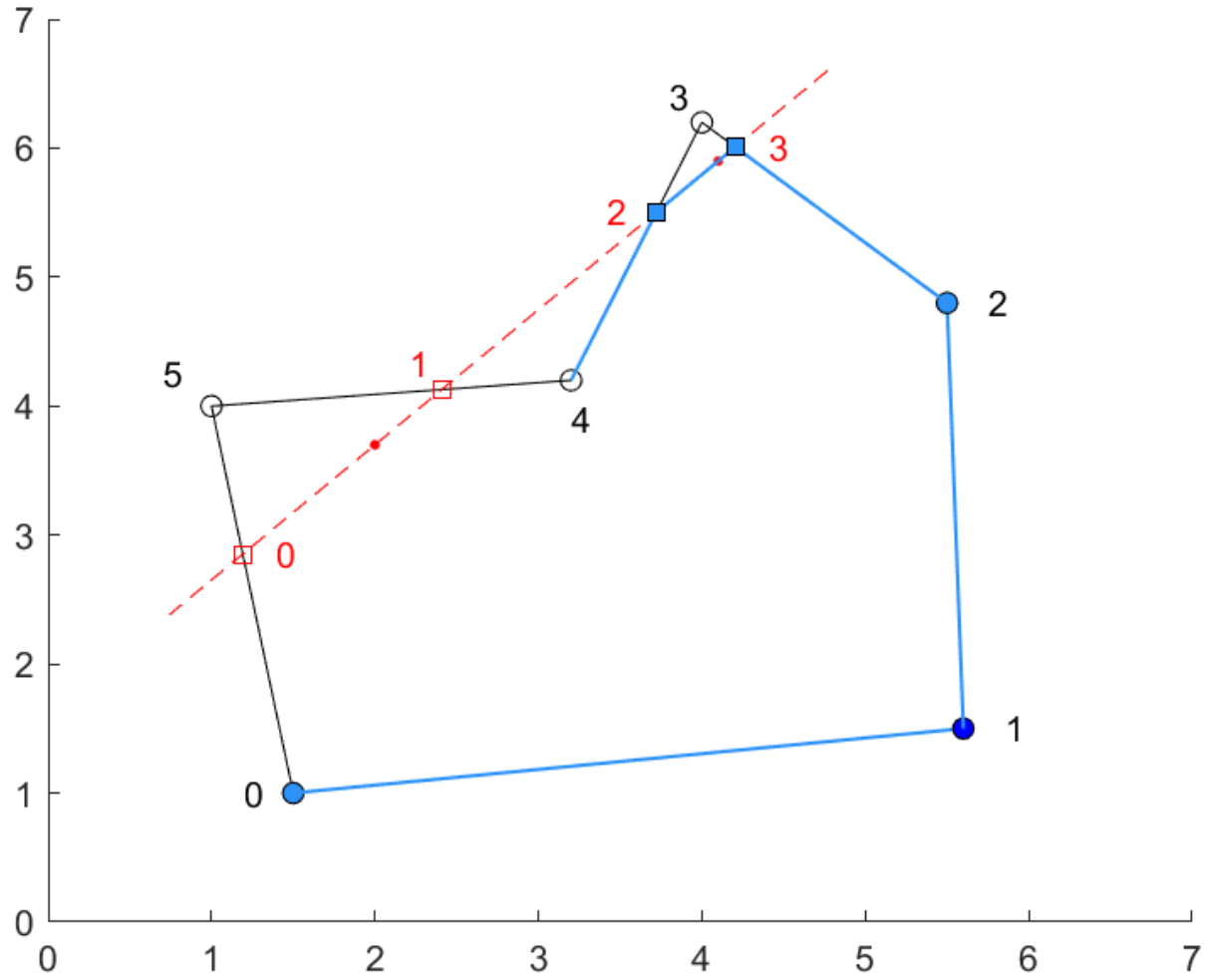


	segToInt
0	-1
1	-1
2	3
3	2
4	1
5	0

	intToSeg
0	5
1	4
2	3
3	2

	found
0	1
1	1
2	1
3	0
4	0
5	0

segToInt[2] = 3, quindi il segmento 2 interseca il punto di intersezione 3. Adesso, siccome 3 è dispari, per rimanere all' interno del poligono, si aggiunge l' id precedente, il 2.
Bisogna ora trovare il punto seguente, in verso antiorario nel poligono, questo è il successivo di intToSeg[2] = 3, e quindi 4.

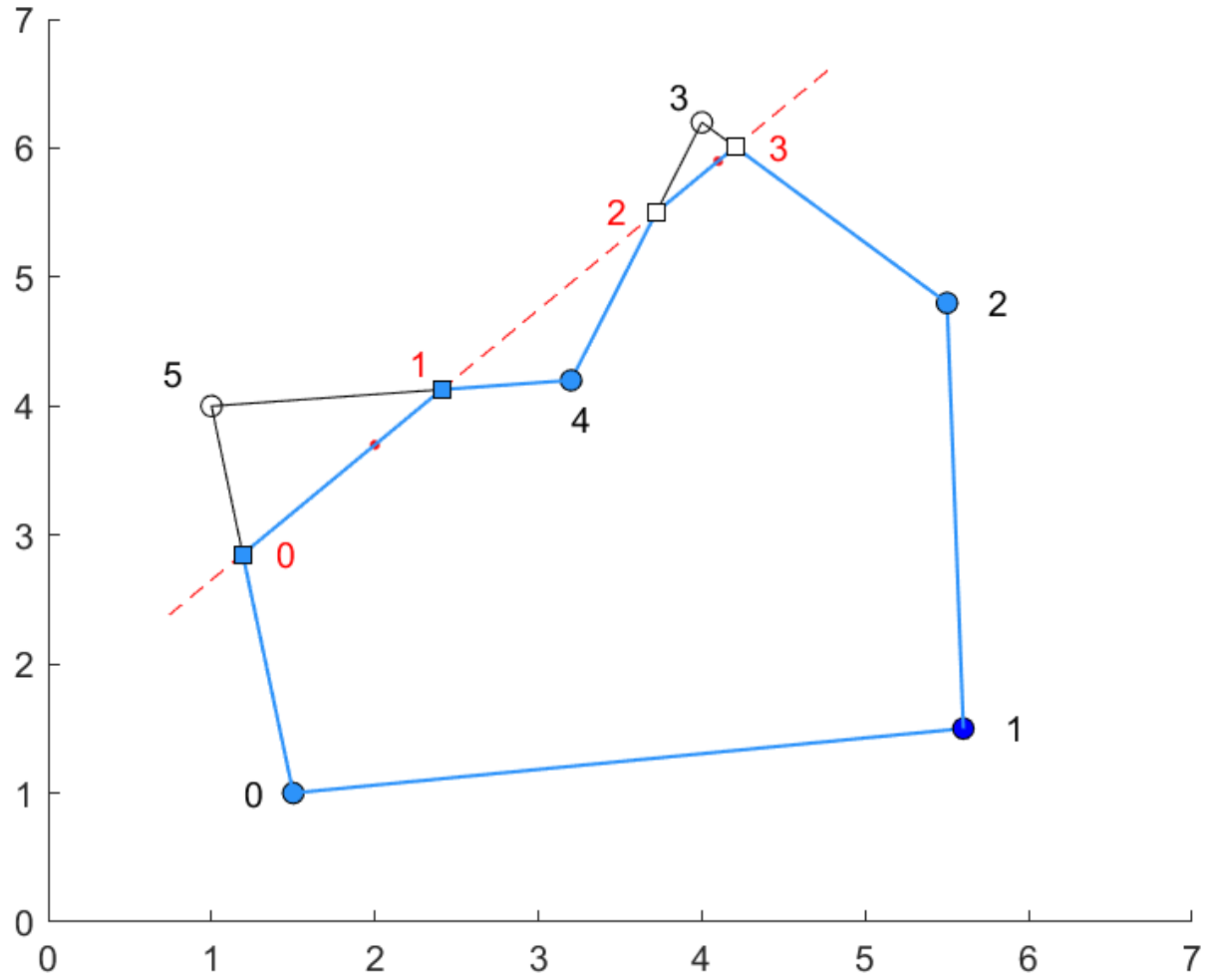


	segToInt
0	-1
1	-1
2	3
3	2
4	1
5	0

	intToSeg
0	5
1	4
2	3
3	2

	found
0	1
1	1
2	1
3	0
4	1
5	0

Sul segmento 4 è presente un intersezione con il punto 1, che è nuovamente dispari, e quindi si aggiunge anche l' intersezione con id 0. intToSeg[0] = 5, il cui successivo, seguendo l' ordine ciclico antiorario del poligono è 0.

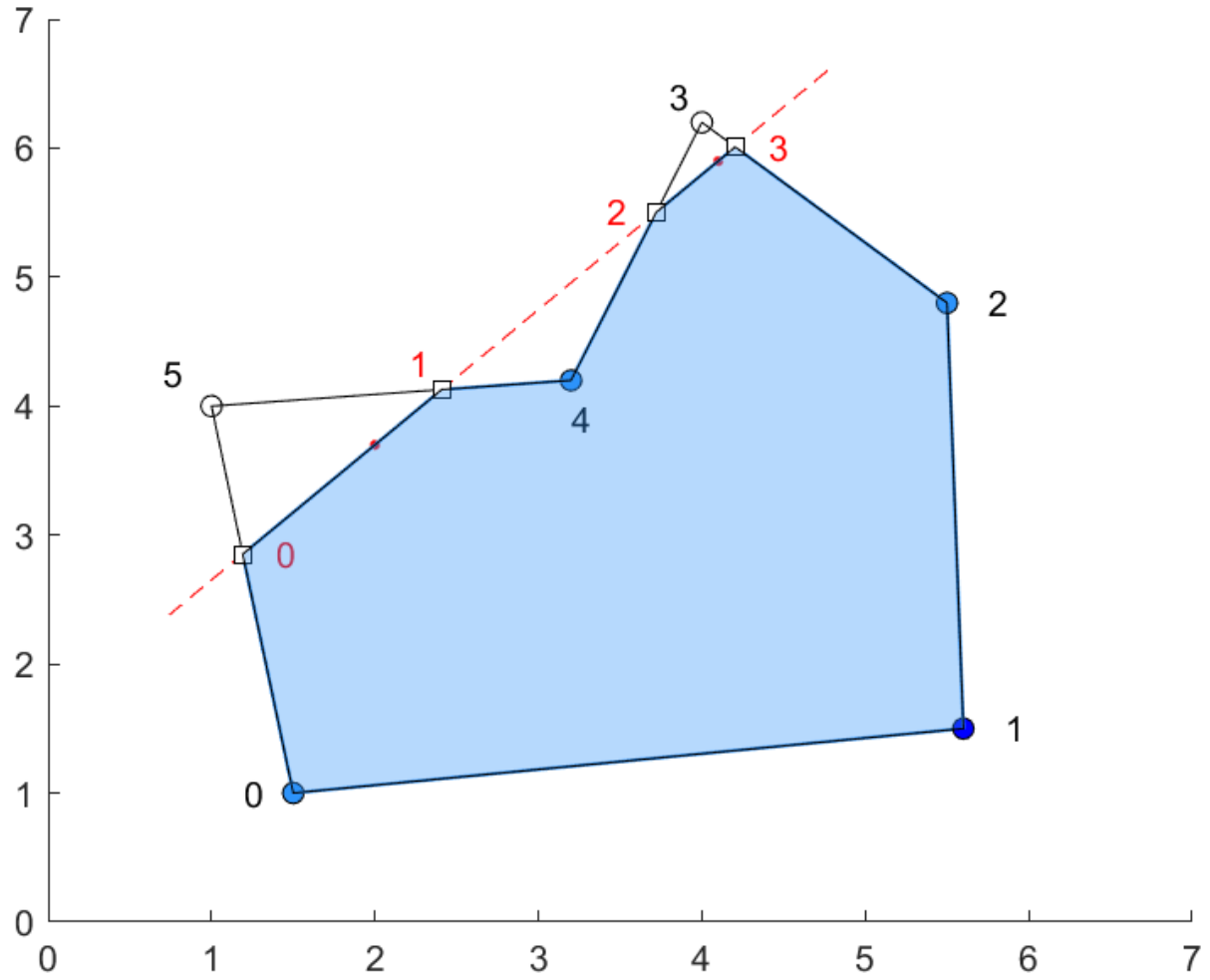


	segToInt
0	-1
1	-1
2	3
3	2
4	1
5	0

	intToSeg
0	5
1	4
2	3
3	2

	found
0	1
1	1
2	1
3	0
4	1
5	0

Si è ritornati ad un punto già trovato, segnalato dal fatto che found[0] = 1, allora si conclude il primo sotto-poligono e l'algoritmo riparte dal punto 3, che non è stato ancora esplorato.

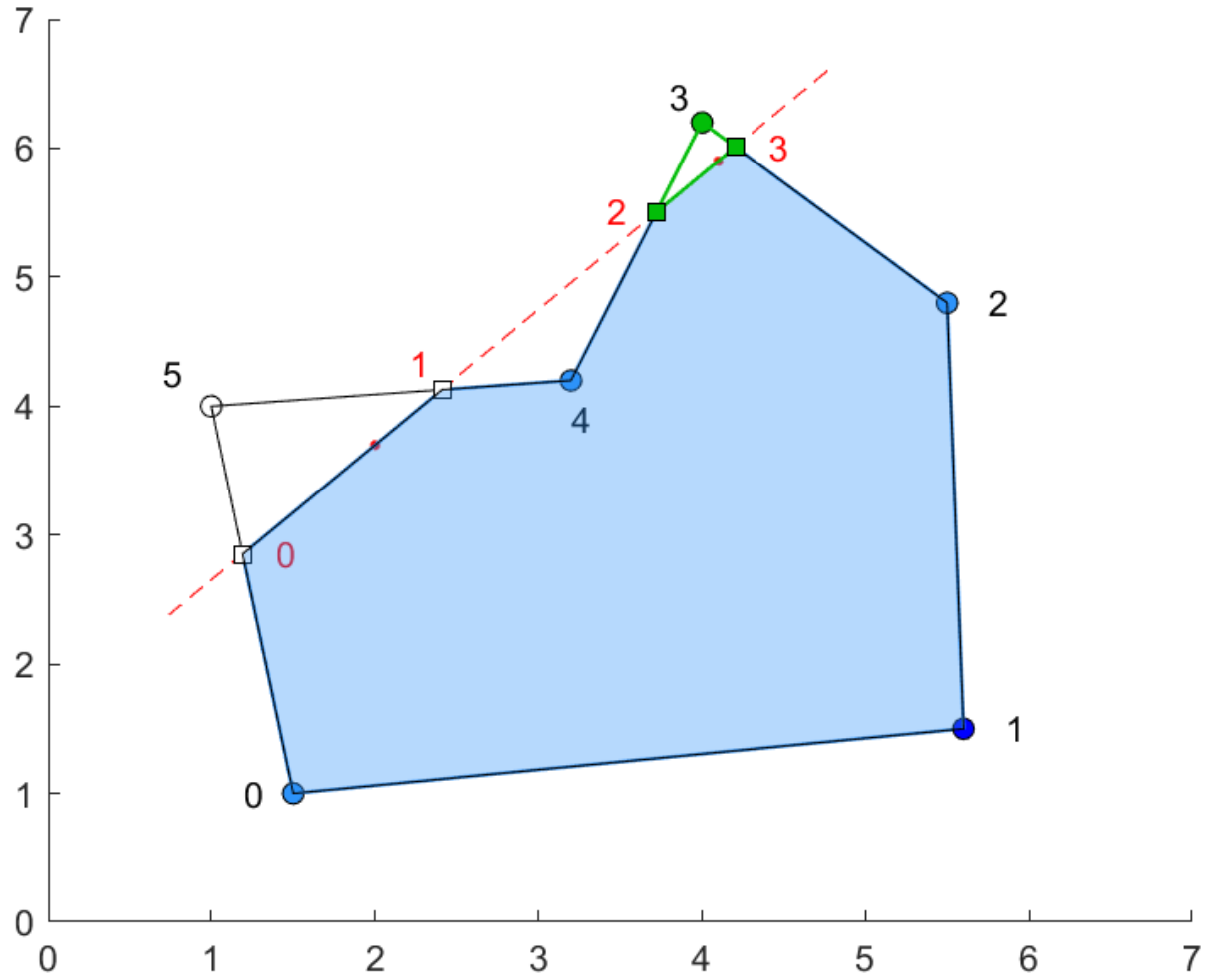


	segToInt
0	-1
1	-1
2	3
3	2
4	1
5	0

	intToSeg
0	5
1	4
2	3
3	2

	found
0	1
1	1
2	1
3	1
4	1
5	0

Da segToInt[3] = 2 si ricava che sul segmento 3 è presente il punto di intersezione 2 che è pari, quindi per rimanere all' interno del poligono si prende il successivo, il 3.
Seguendo il verso antiorario si ritorna al punto di partenza.

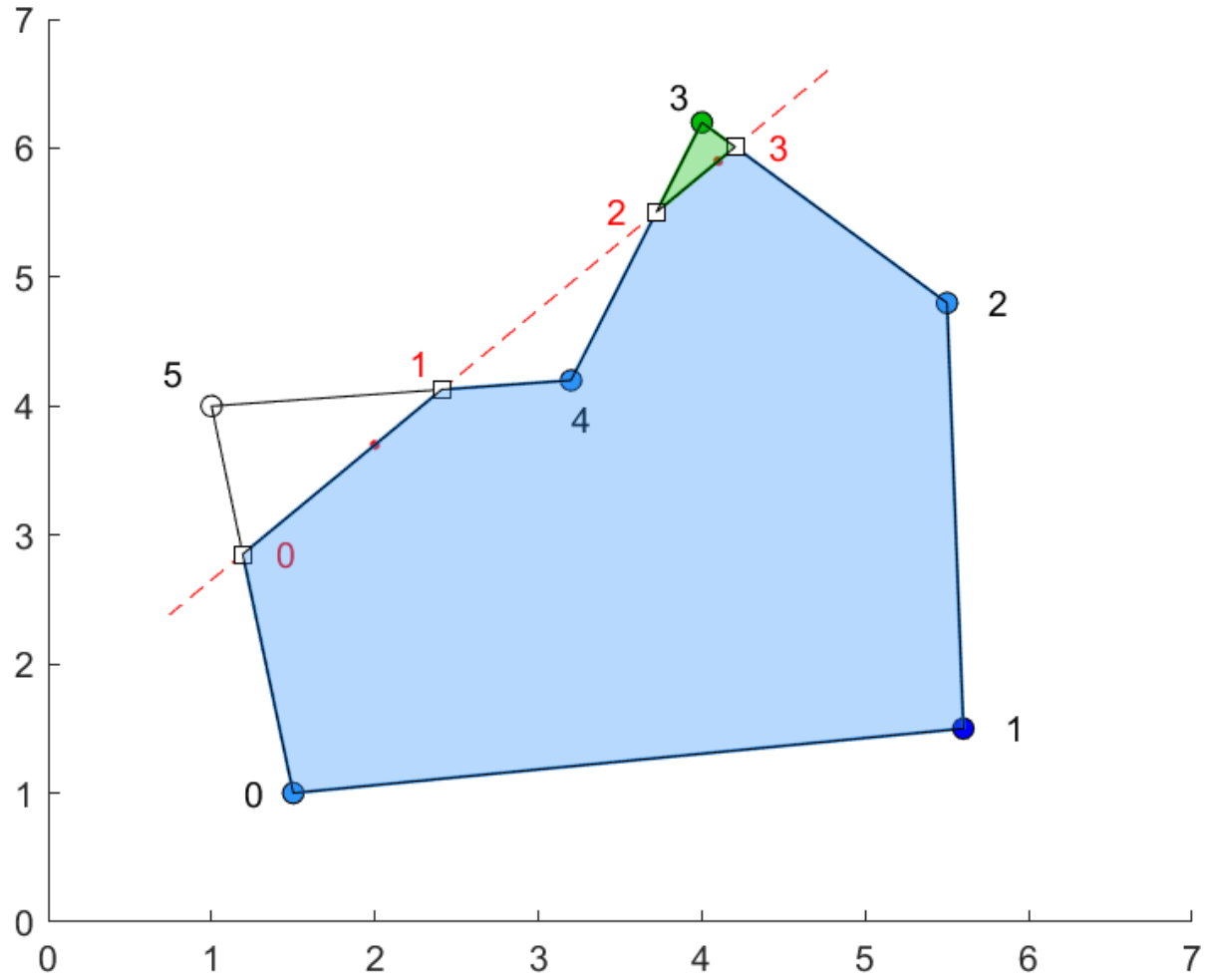


	segToInt
0	-1
1	-1
2	3
3	2
4	1
5	0

	intToSeg
0	5
1	4
2	3
3	2

	found
0	1
1	1
2	1
3	1
4	1
5	0

Found[3] = 1, il punto 3 è già stato visitato, si conclude quindi il secondo sotto-poligono.

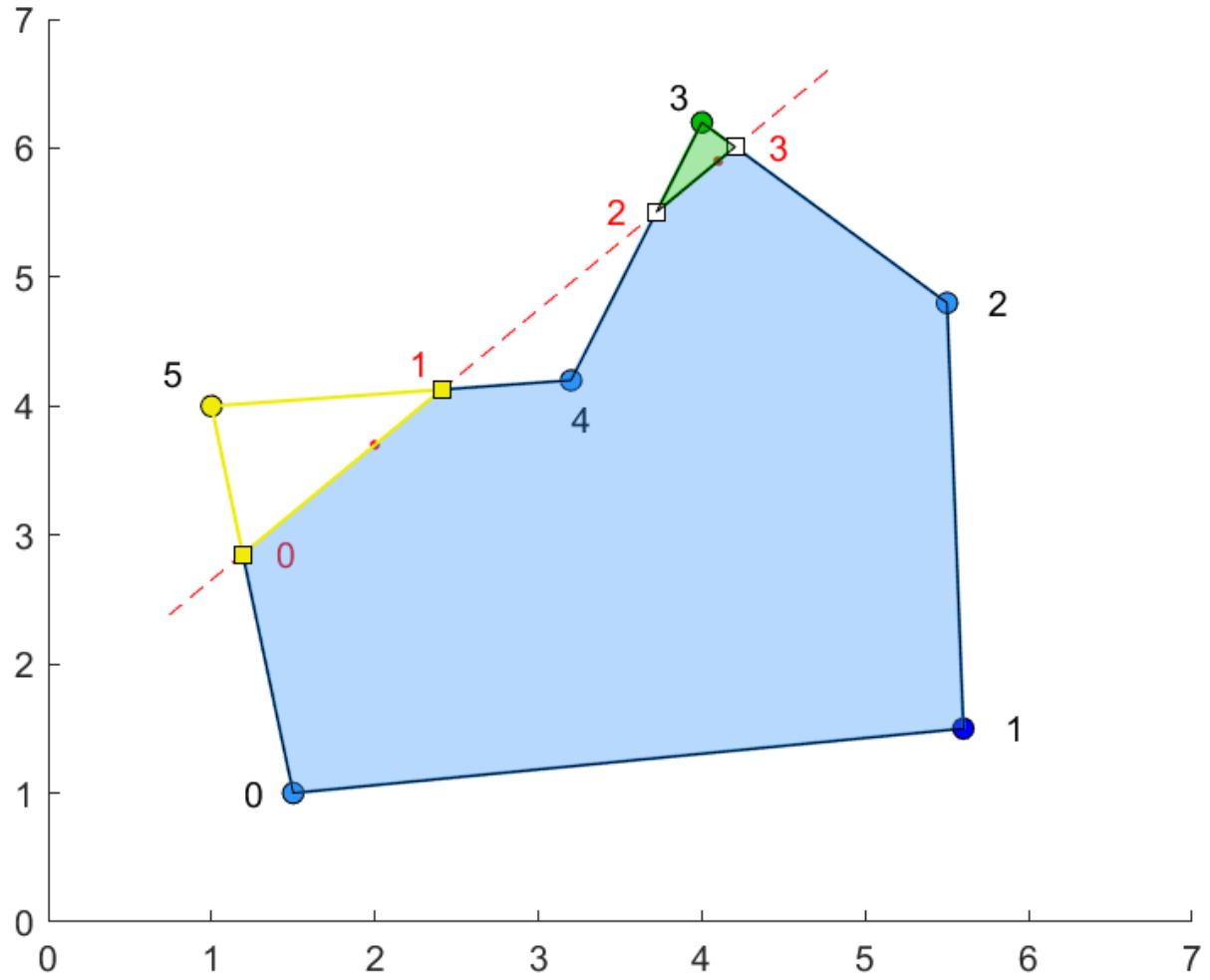


	segToInt
0	-1
1	-1
2	3
3	2
4	1
5	0

	intToSeg
0	5
1	4
2	3
3	2

	found
0	1
1	1
2	1
3	1
4	1
5	1

Si esplora il punto 5 e si arriva al punto di intersezione pari 0, quindi si va al successivo 1 per poi tornare all' inizio.

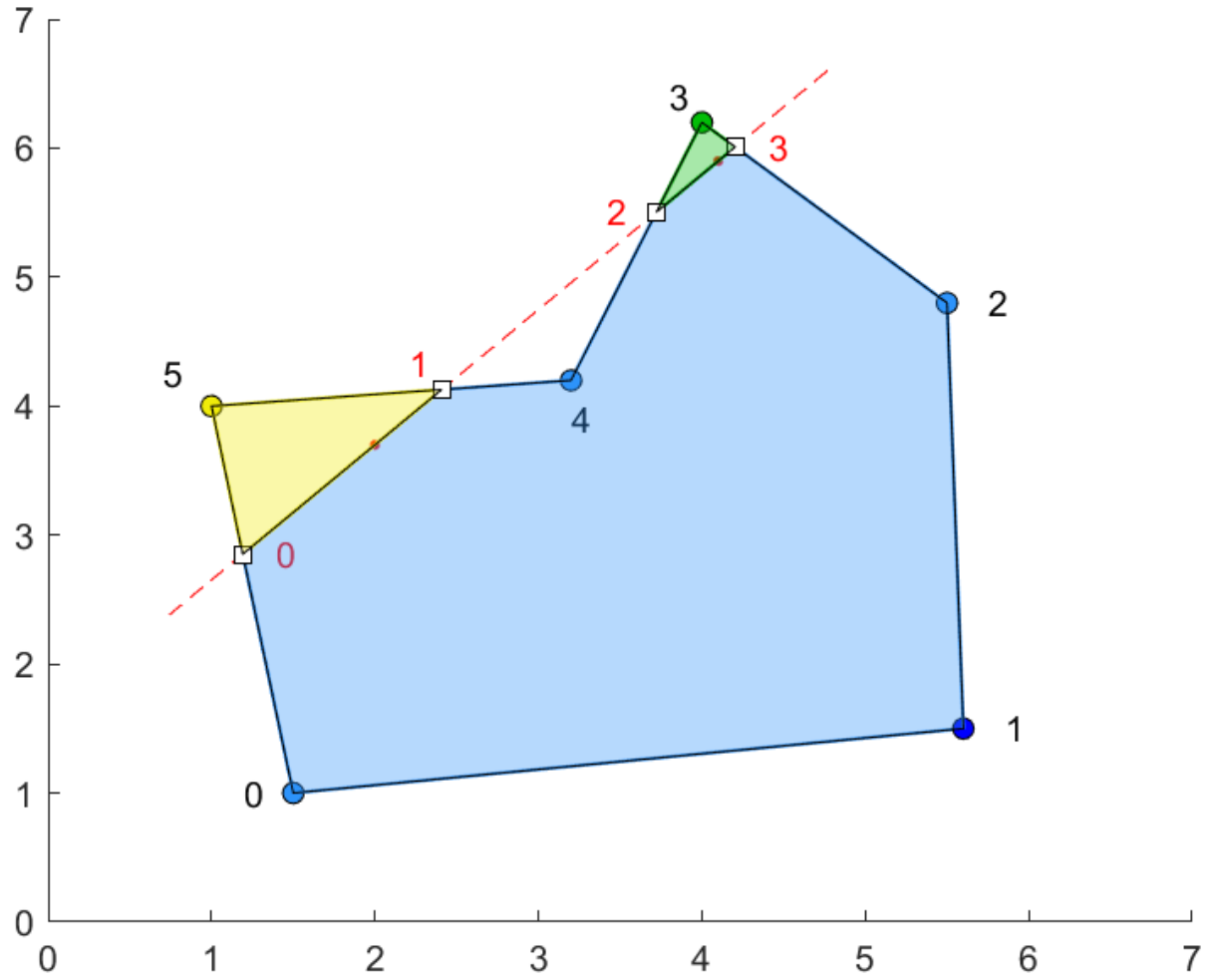


	segToInt
0	-1
1	-1
2	3
3	2
4	1
5	0

	intToSeg
0	5
1	4
2	3
3	2

	found
0	1
1	1
2	1
3	1
4	1
5	1

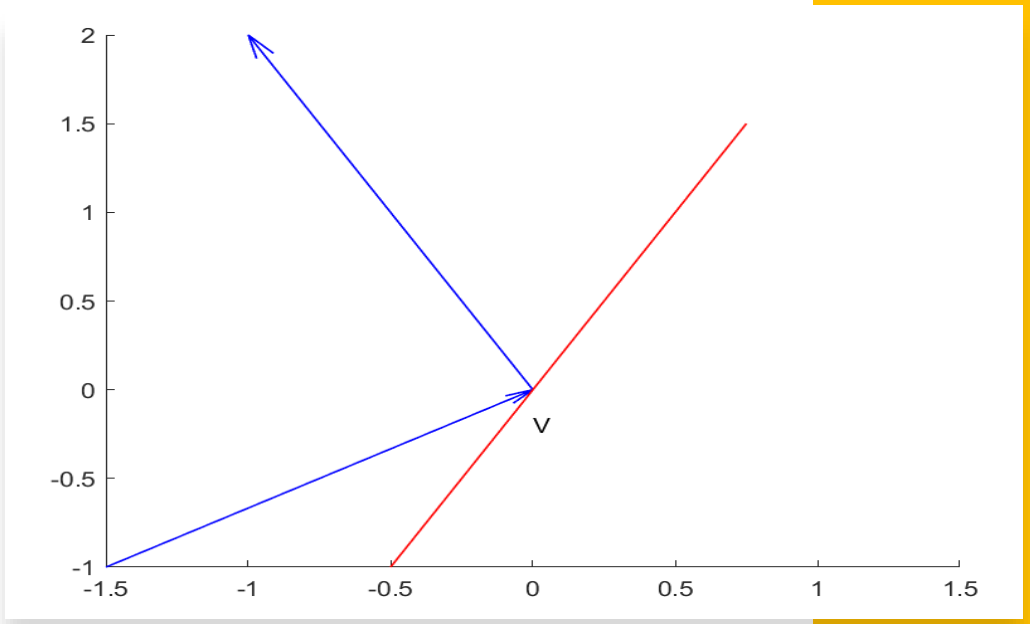
Abbiamo così trovato tutti i sotto-poligoni, infatti found ha tutti valori true e l'algoritmo termina.

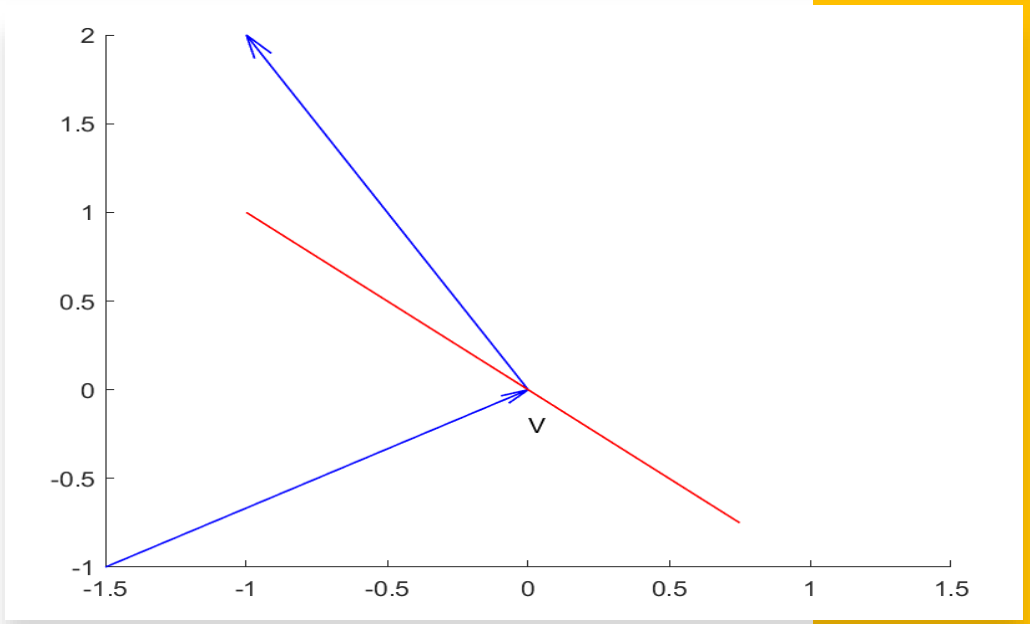


Intersezione vertice 1

Avere intersezioni in numero pari è comodo perché permette di avere coppie di segmenti interni al poligono che si formano sulla retta di intersezione. Studiamo quindi il caso di intersezione sul vertice con questo obbiettivo e notando che ci sono due tipi di intersezione da studiare in modo diverso.

1. Sfiora il vertice ed è come se il segmento entrasse e subito uscisse, quindi il vertice è da considerare un intersezione doppia.
2. Attraversa il vertice per entrare o uscire ed allora consideriamo una singola intersezione.

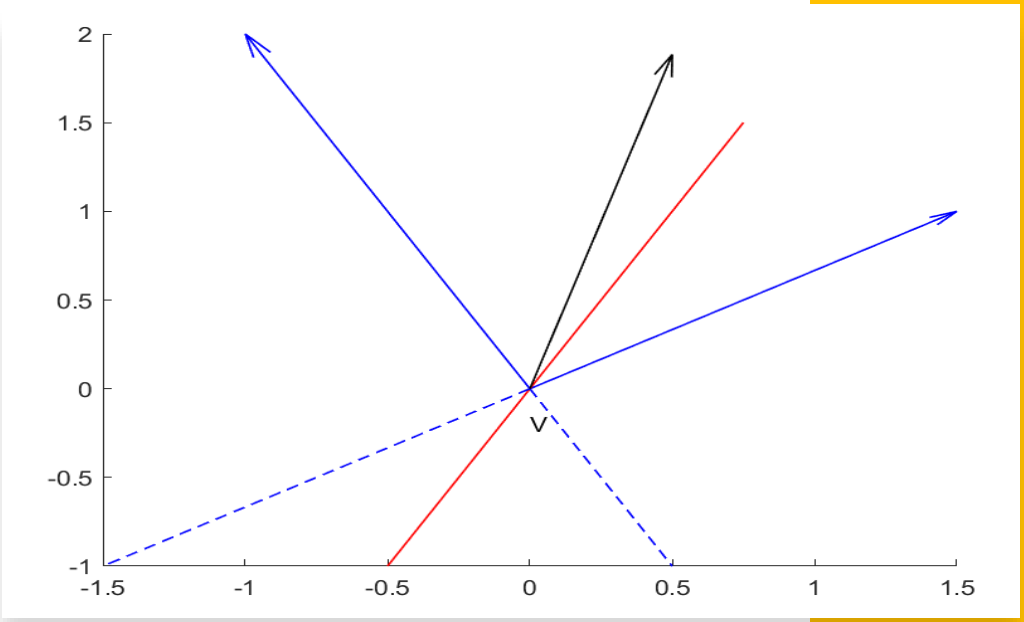


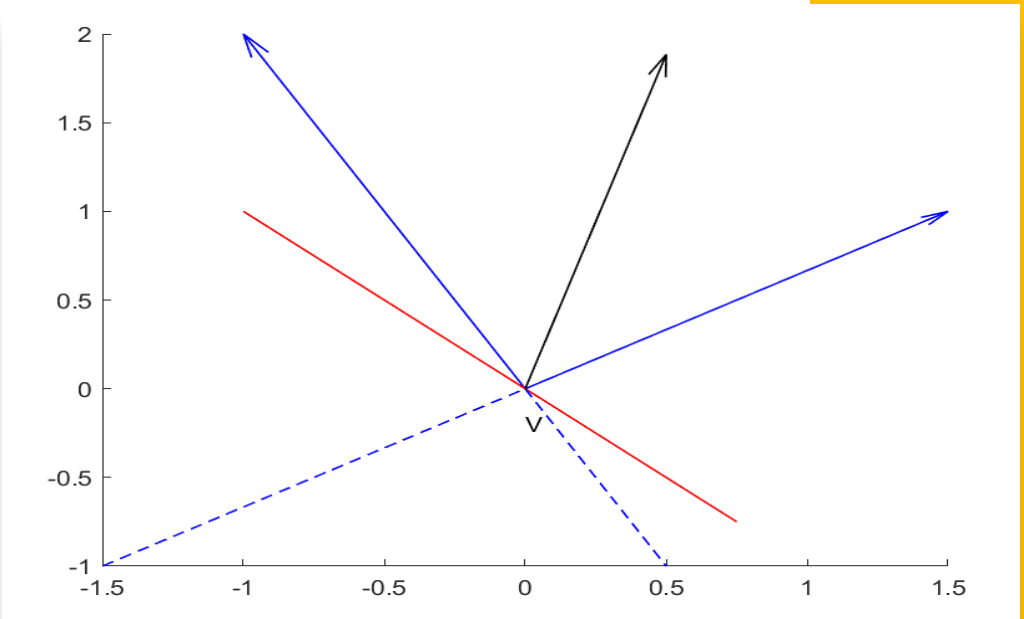


Se consideriamo i lati come vettori applicati all' origine notiamo che siamo nel primo caso se il segmento è all' interno della loro combinazione conica, nel secondo altrimenti.
Utilizzando la bisettrice dei segmenti per calcolare gli angoli da questa si possono distinguere i due casi.

In questo modo possiamo ricondurci, con semplici controlli per evitare di inserire più volte punti con le stesse coordinate, all' algoritmo precedente.

Tuttavia nei sotto-poligoni questi punti possono essere sia vertici che intersezioni, quindi formalmente distinti nonostante rappresentino lo stesso punto. Per questo motivo è stato introdotto un attributo label per i punti in modo da associare univocamente un intero alle coordinate.



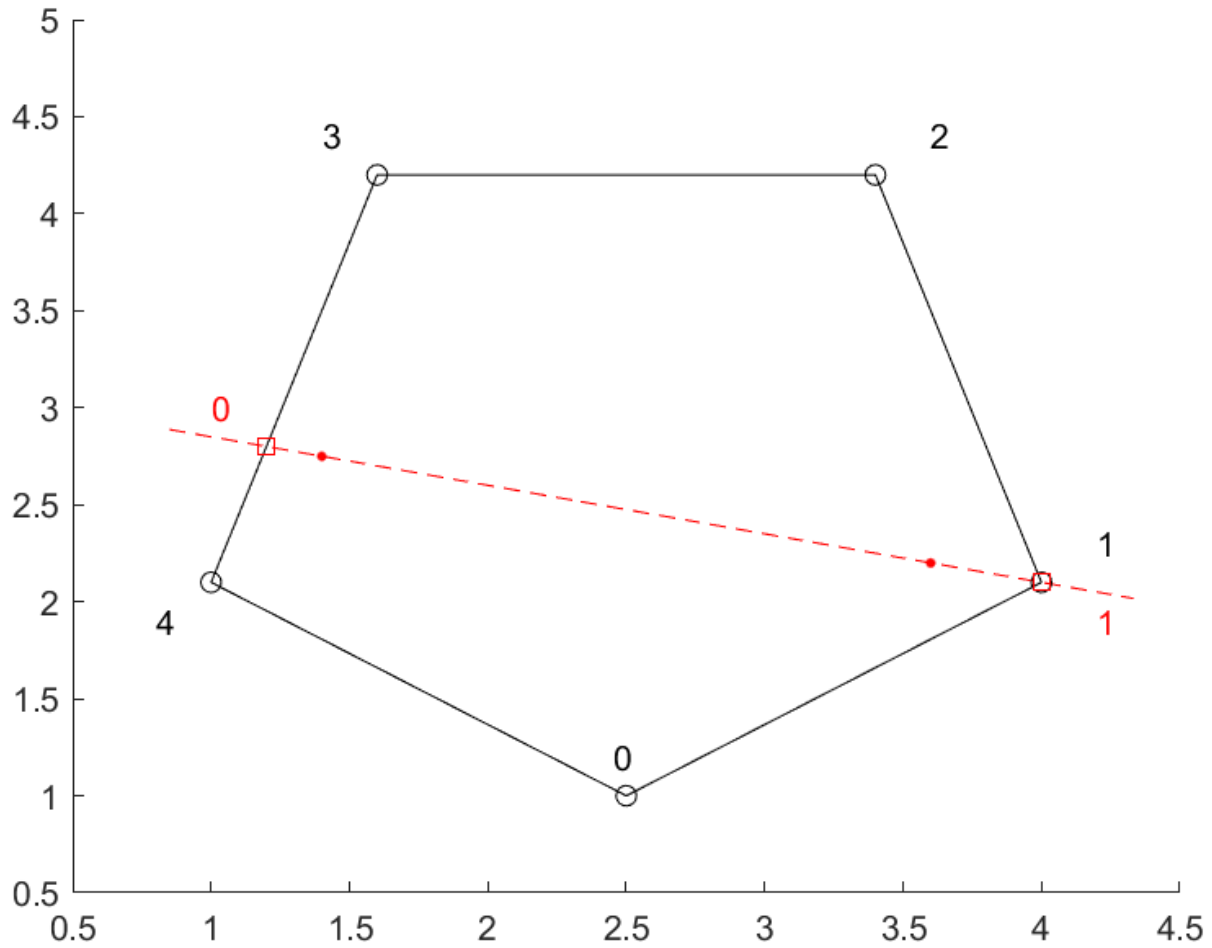


	segToInt
0	1
1	-1
2	-1
3	0
4	-1

	intToSeg
0	3
1	0

	found
0	0
1	0
2	0
3	0
4	0

Notiamo che il numero di intersezioni è pari, infatti l intersezione 1 sul vertice è stata considerata una sola volta.
Inoltre intToSeg[1] = 0 e non 1 perché l' intersezione con i segmenti viene fatta in verso antiorario.

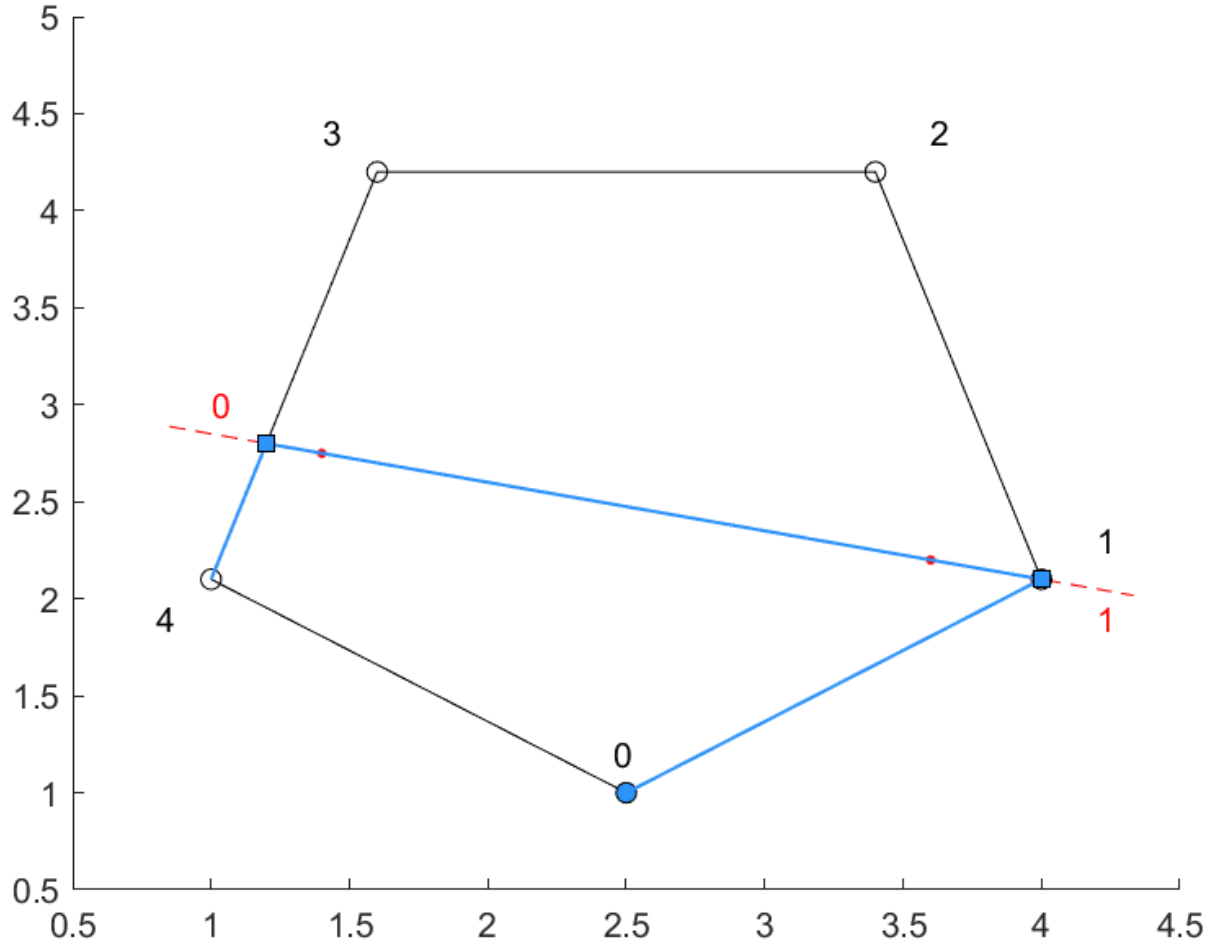


	segToInt
0	1
1	-1
2	-1
3	0
4	-1

	intToSeg
0	3
1	0

	found
0	1
1	0
2	0
3	0
4	0

Nonostante l'intersezione sia sul vertice l'algoritmo procede come al solito aggiungendo al sotto-poligono l'intersezione 1 e poi la 0. Tuttavia a priori la label dell'intersezione 1 viene impostata uguale a quella del vertice 1, in modo da considerare i due punti coincidenti.

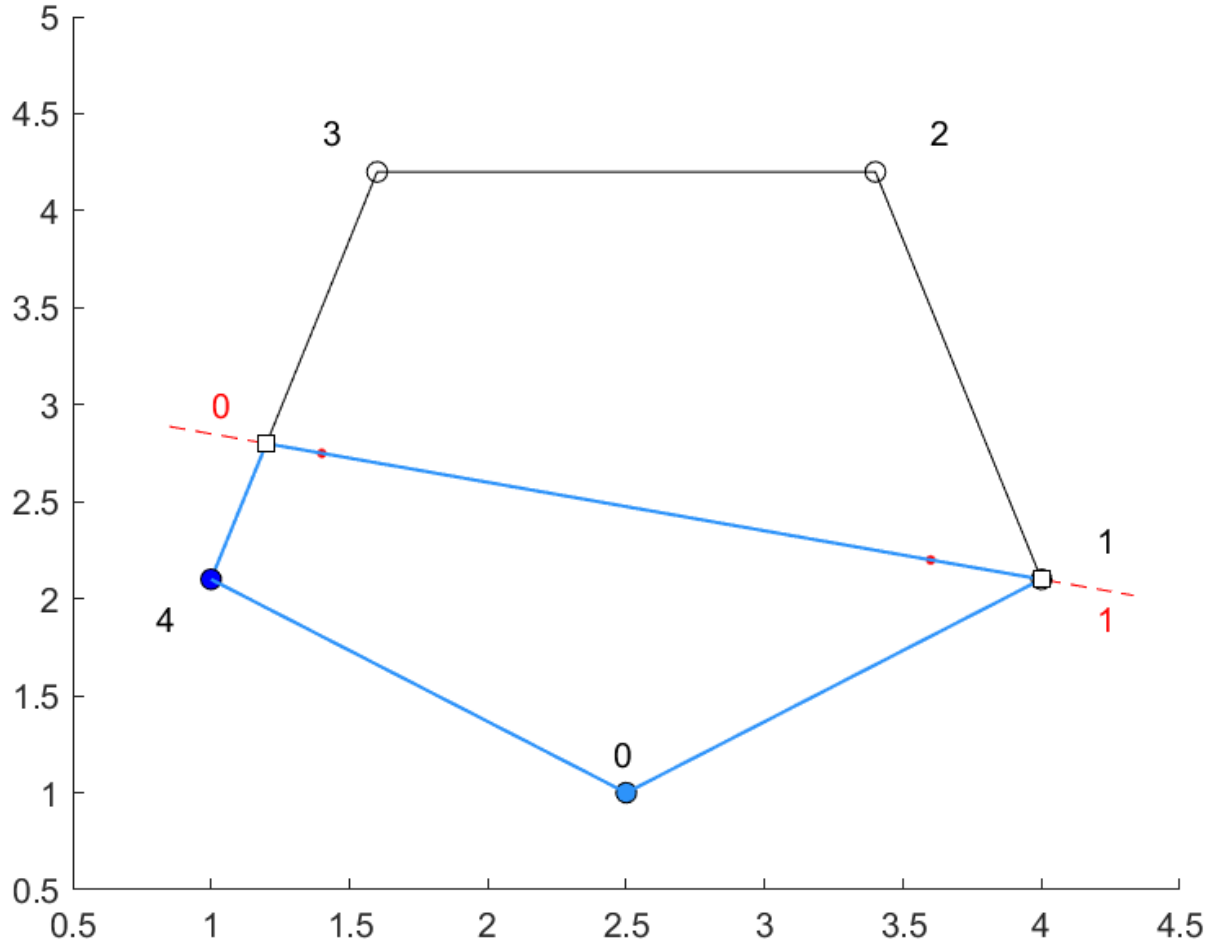


	segToInt
0	1
1	-1
2	-1
3	0
4	-1

	intToSeg
0	3
1	0

	found
0	1
1	0
2	0
3	0
4	1

segToInt[4] = -1, l’ algoritmo procede al prossimo vertice ritrovando 0.

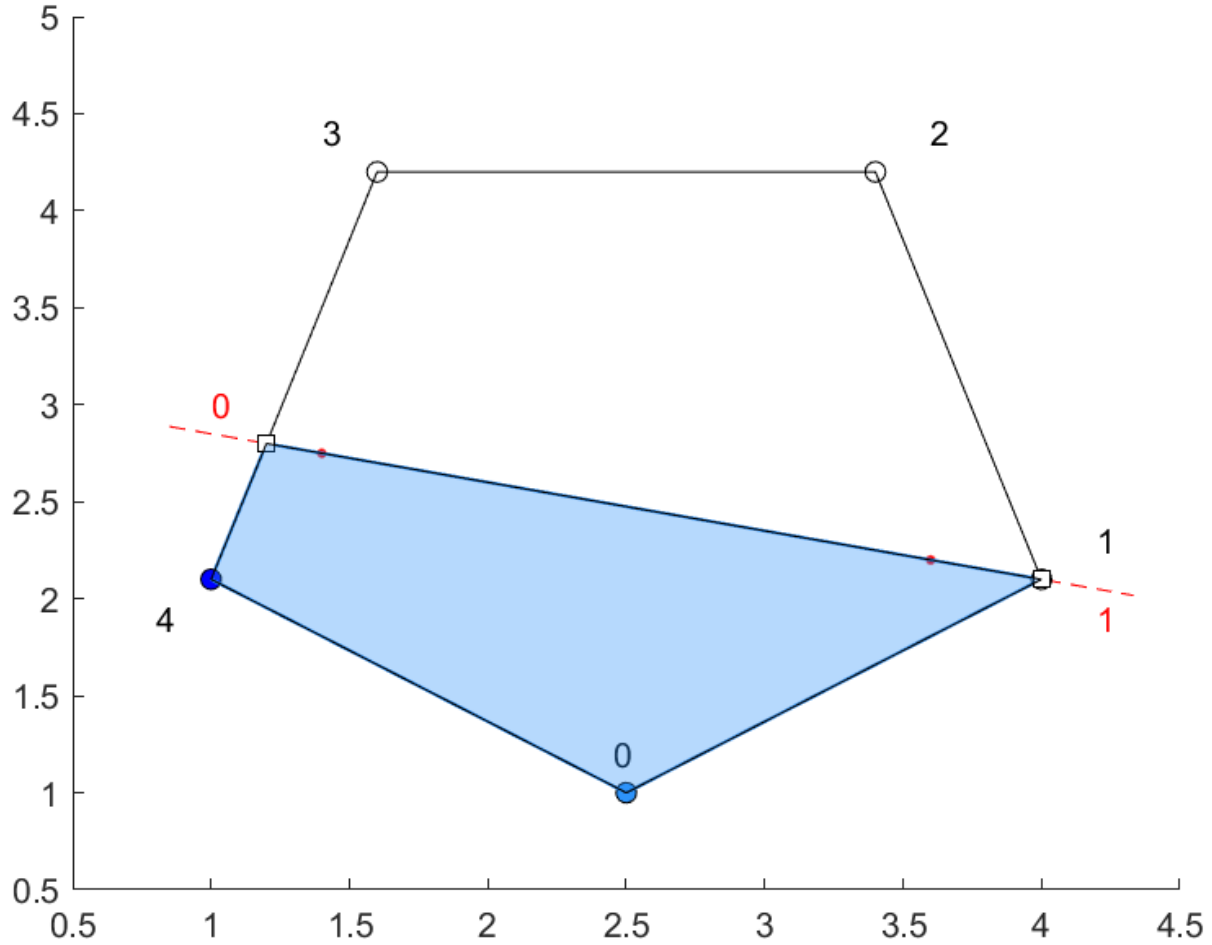


	segToInt
0	1
1	-1
2	-1
3	0
4	-1

	intToSeg
0	3
1	0

	found
0	1
1	0
2	0
3	0
4	1

L’algoritmo torna ad un punto già trovato, infatti found[0] = 1 e termina il primo sotto-poligono.

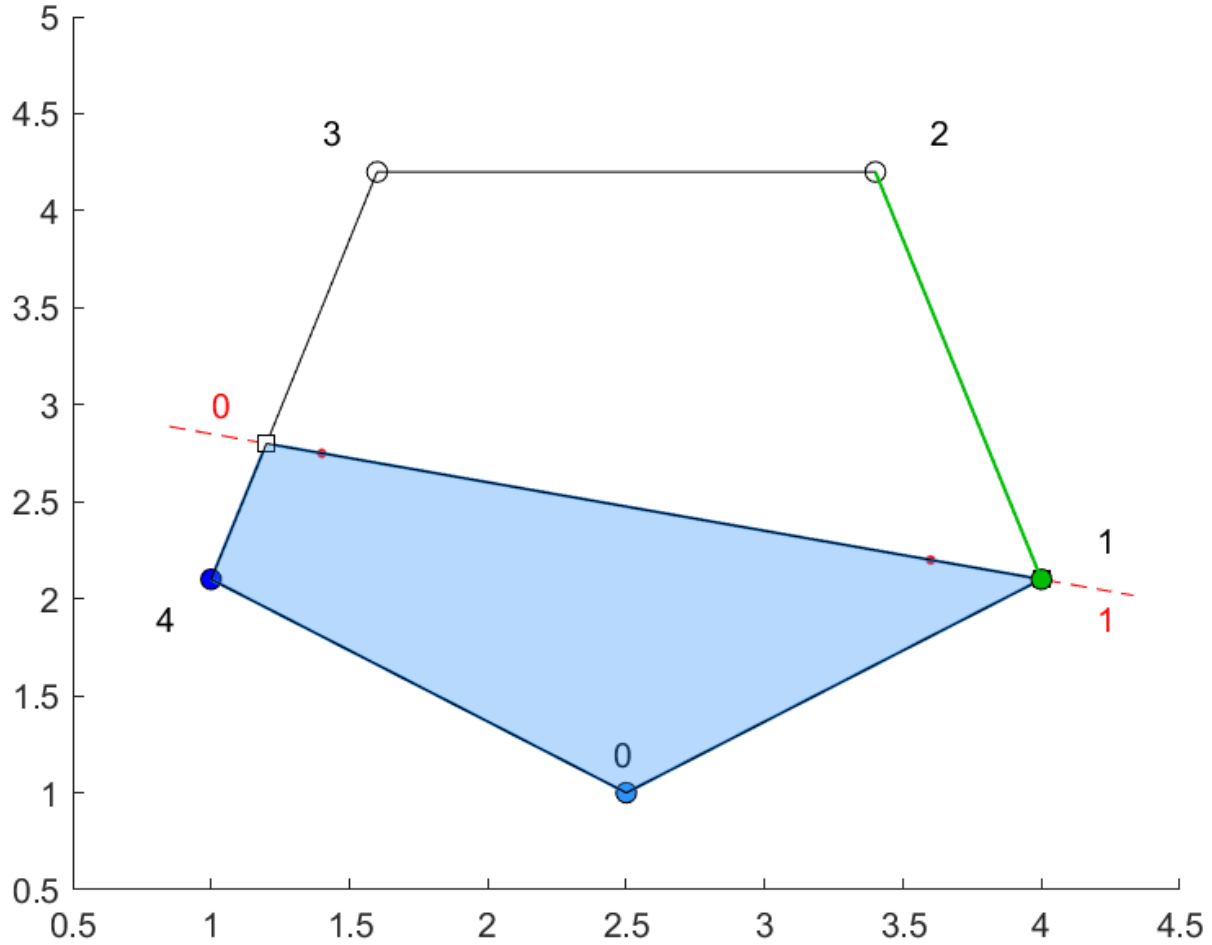


	segToInt
0	1
1	-1
2	-1
3	0
4	-1

	intToSeg
0	3
1	0

	found
0	1
1	1
2	0
3	0
4	1

Riparte dal vertice successivo che formalmente non è ancora stato esplorato, quindi 1.
Infatti il poligono precedente ha aggiunto lo stesso punto ma preso dai punti di intersezione.

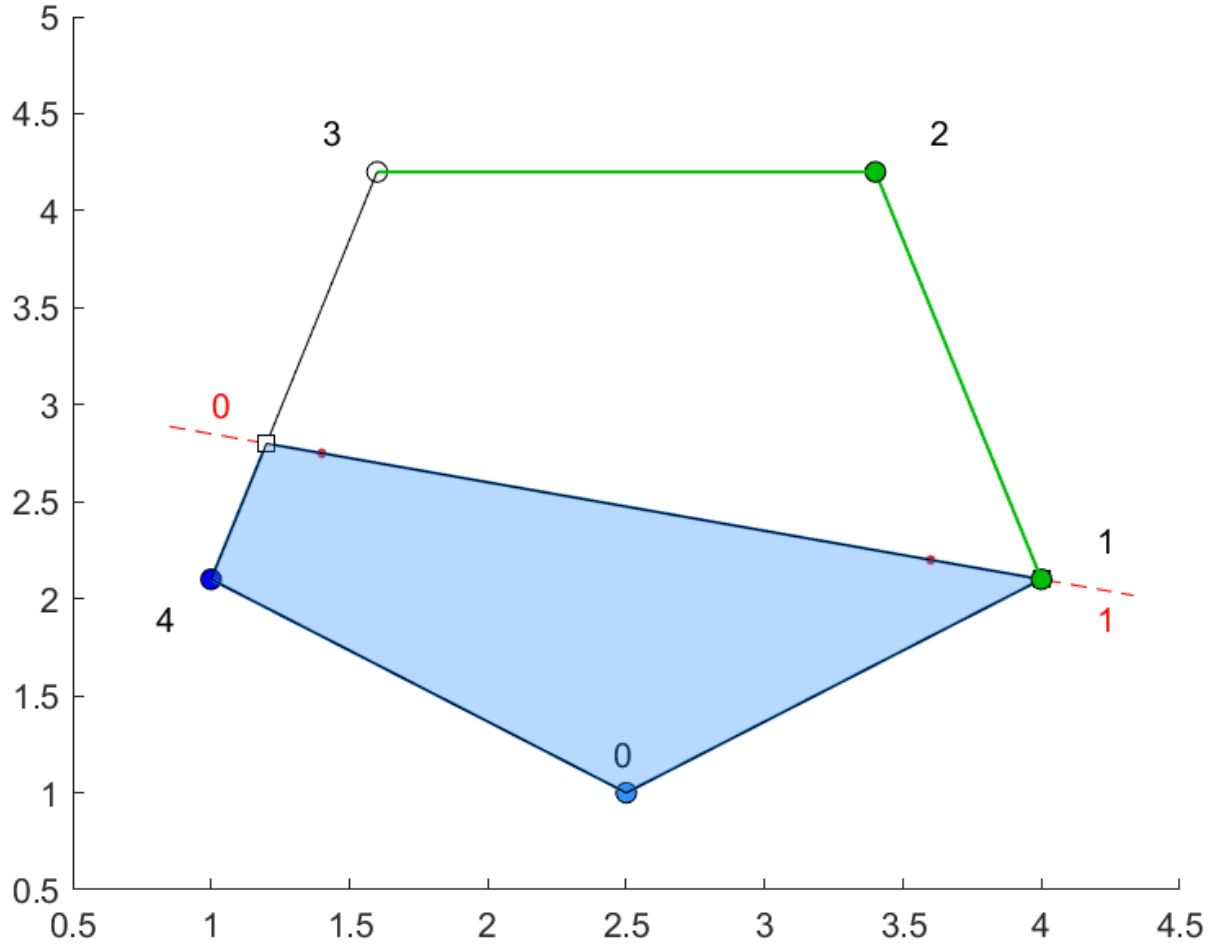


	segToInt
0	1
1	-1
2	-1
3	0
4	-1

	intToSeg
0	3
1	0

	found
0	1
1	1
2	1
3	0
4	1

Lungo il segmento 2 non c'è intersezione e si procede verso il vertice 3.

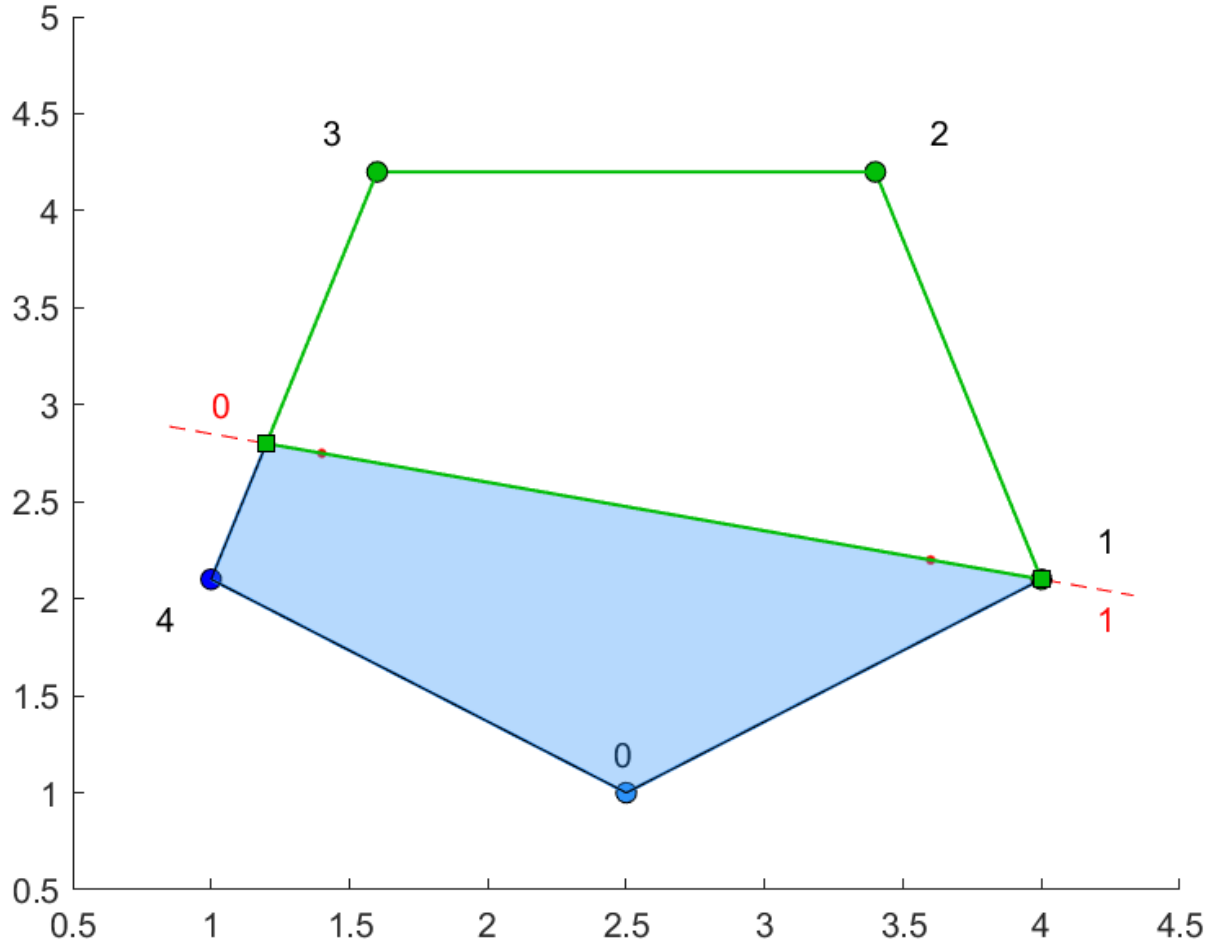


	segToInt
0	1
1	-1
2	-1
3	0
4	-1

	intToSeg
0	3
1	0

	found
0	1
1	1
2	1
3	1
4	1

segToInt[3] = 0, l'algoritmo aggiunge l'intersezione 0 e procede verso l' intersezione 1. Con un controllo si verifica se il punto che si vuole aggiungere coincida con il successivo. Se succede non si aggiunge tale punto e si continua. In questo particolare caso si torna al vertice di partenza.

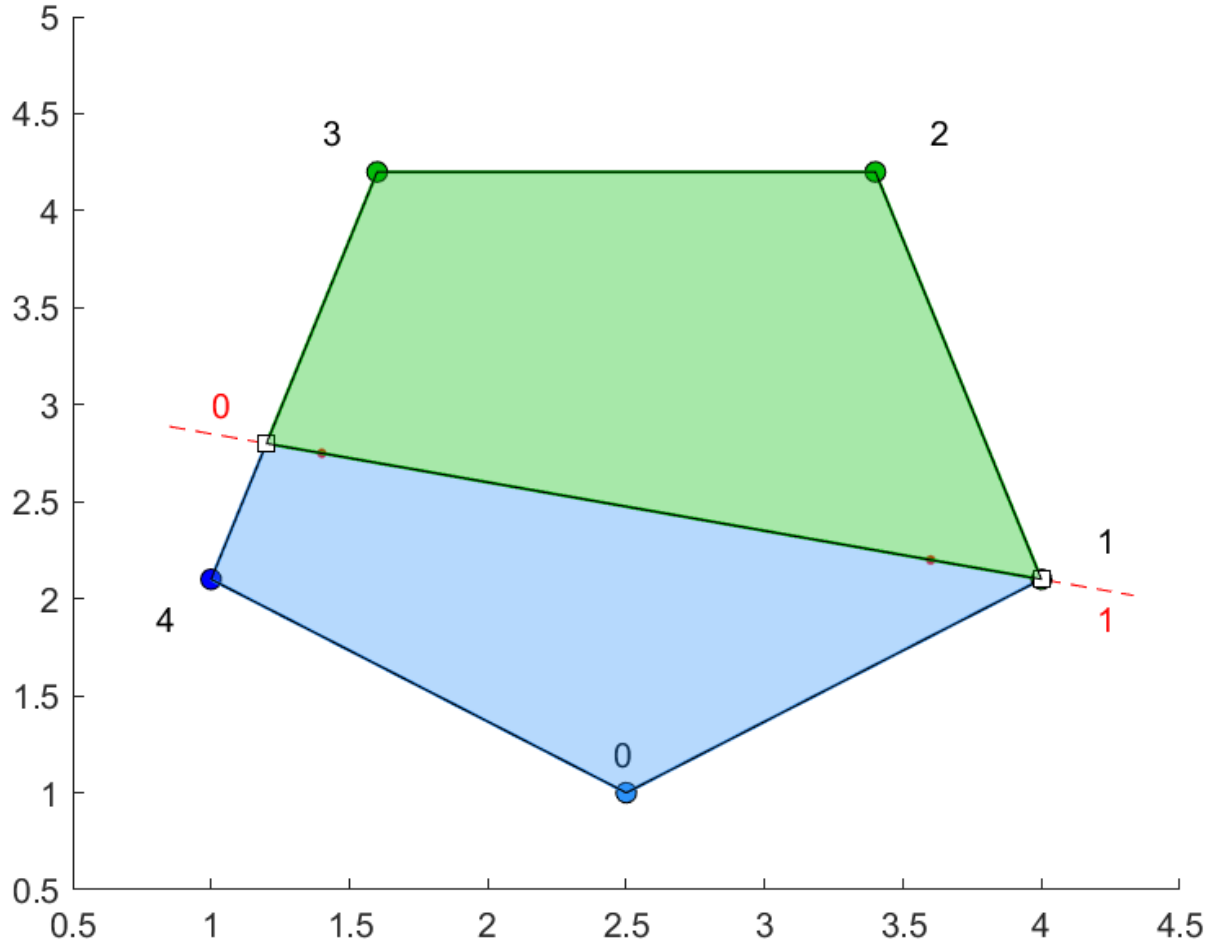


	segToInt
0	1
1	-1
2	-1
3	0
4	-1

	intToSeg
0	3
1	0

	found
0	1
1	1
2	1
3	1
4	1

Tutto il vettore found ha valore 1 ed abbiamo allora esplorato tutti i vertici trovando tutti i sotto-poligoni.

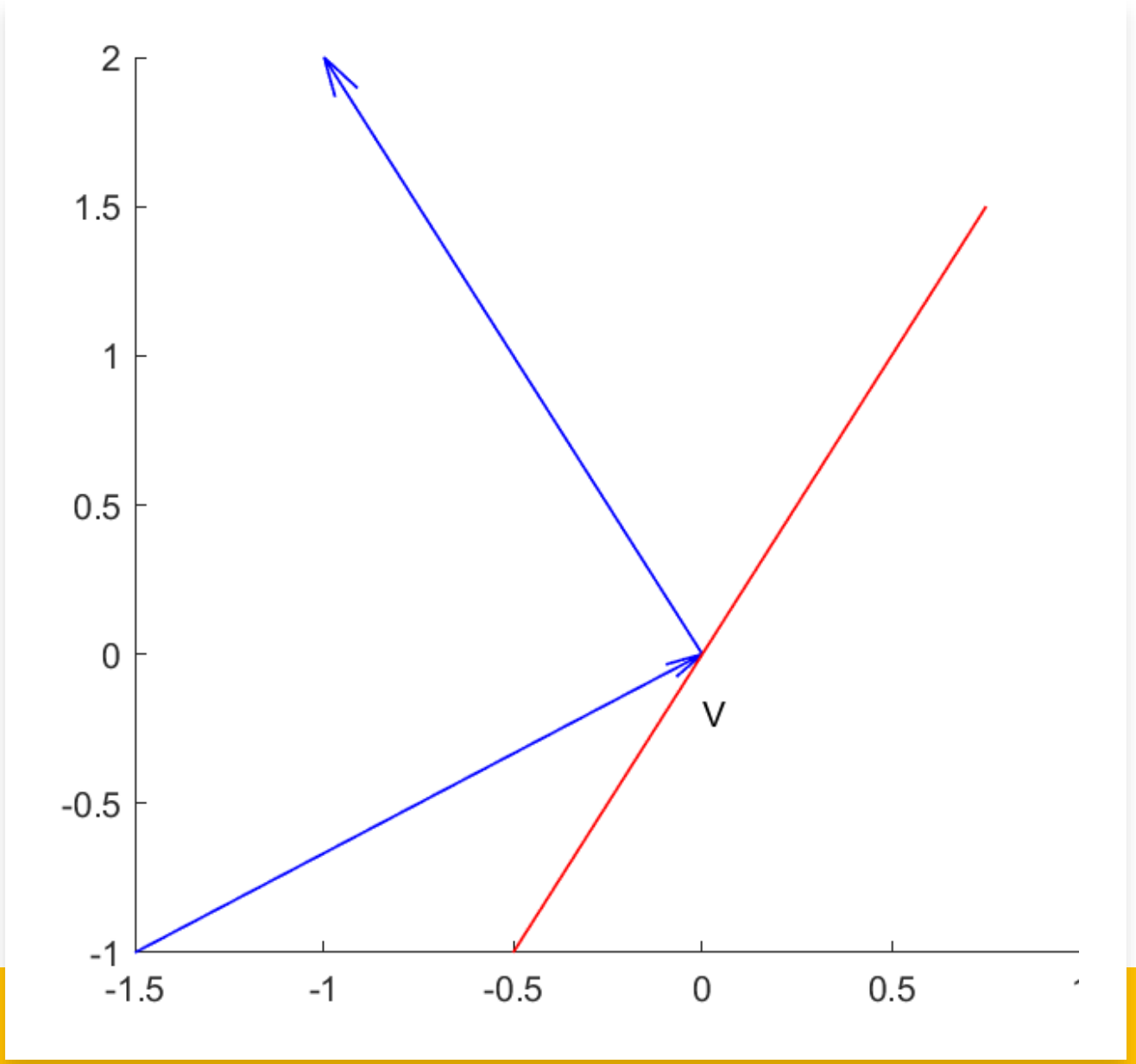


Intersezione vertice 2

Mostriamo adesso l’altro di intersezione e vediamo con due ulteriori esempi i due casi possibili.

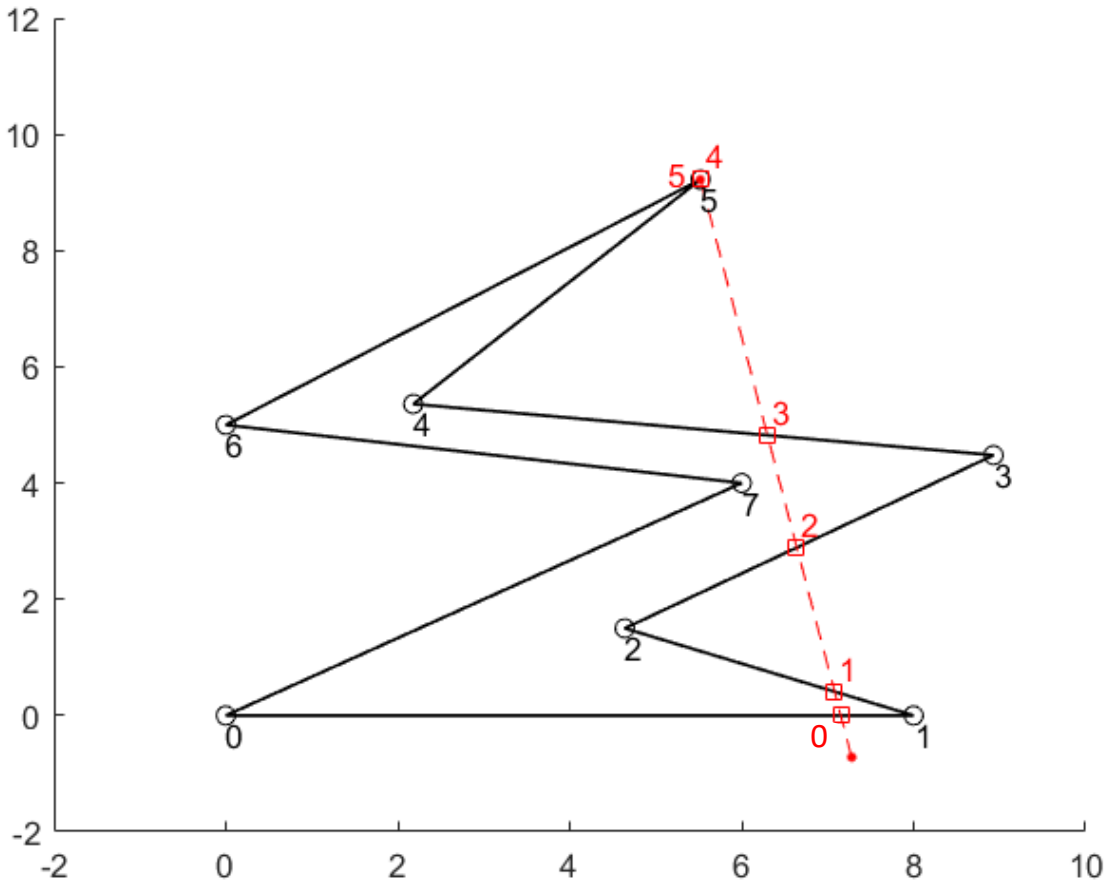
Notiamo che in entrambi i punti di intersezione sono in numero pari proprio perchè il vertice di intersezione è stato aggiunto due volte.

Le immagine sono prodotte dal metodo showPolygon su uno script Matlab.



	segToInt
0	0
1	1
2	2
3	3
4	4
5	5
6	-1
7	-1

	intToSeg
0	0
1	1
2	2
3	3
4	4
5	5

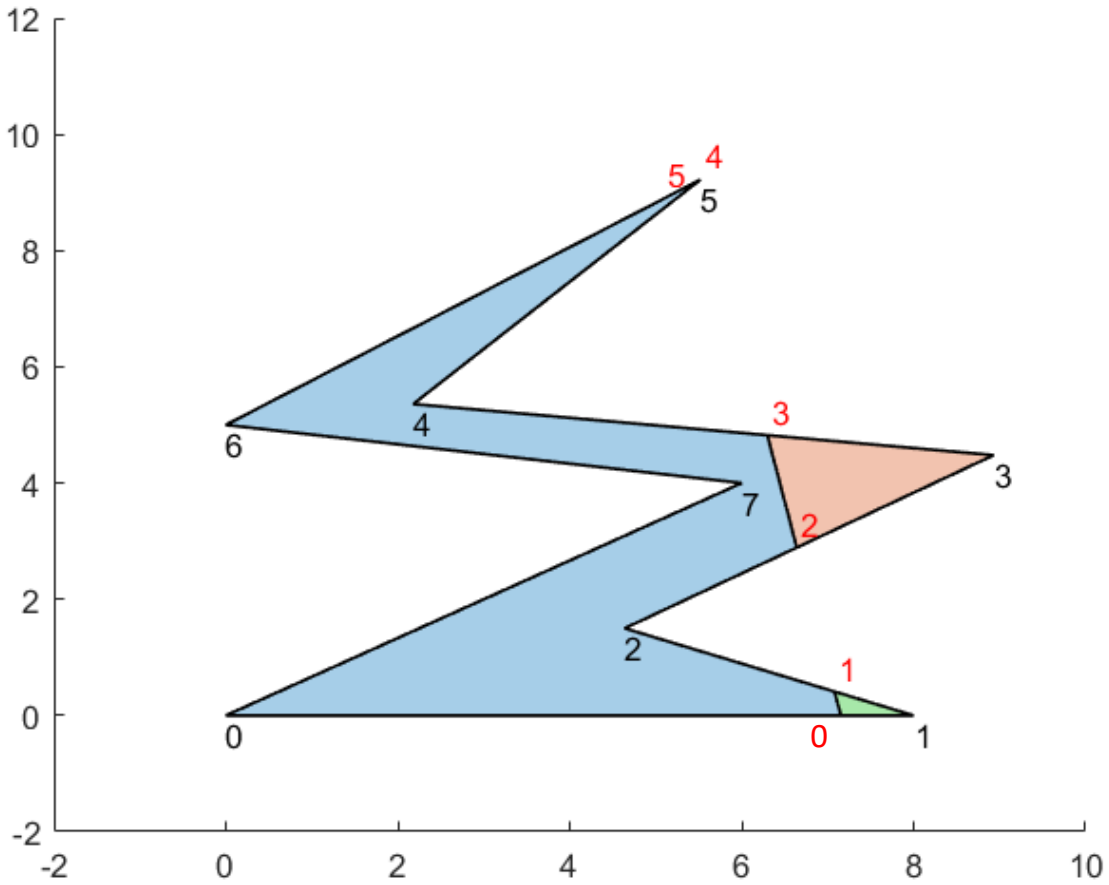


Il vertice 5 è isolato dalle due intersezioni con coincidono con il vertice stesso.
L' algoritmo quindi, procedendo in verso antiorario, aggiunge il punto di intersezione 4, coincidente con il vertice che viene segnato come esplorato, e passa direttamente al punto seguente senza aggiungere l intersezione 5 perché coincidono.

	segToInt
0	0
1	1
2	2
3	3
4	4
5	5
6	-1
7	-1

	intToSeg
0	0
1	1
2	2
3	3
4	4
5	5

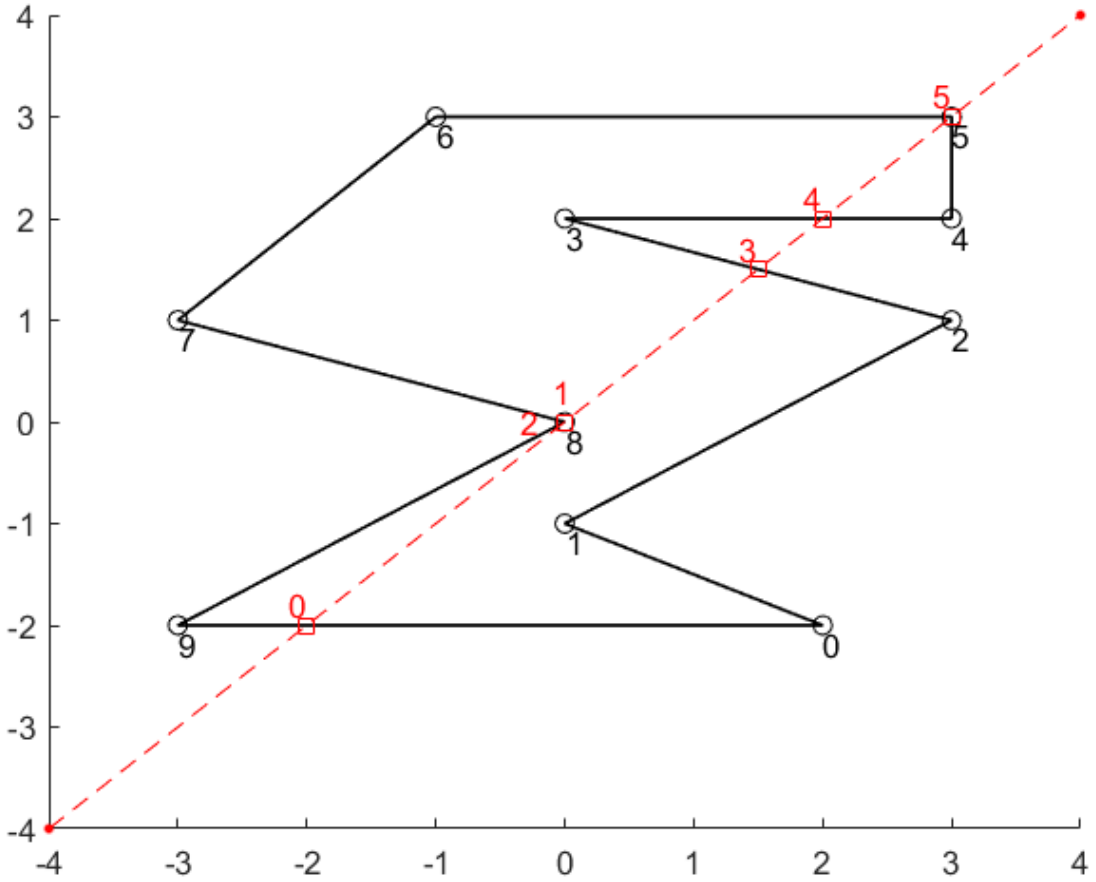
Notiamo che in questo caso nel vertice 5 il ‘segmento’ di intersezione interno (che congiunge 4 con 5) è formato da punti coincidenti e questo differenzia questo caso.



	segToInt
0	-1
1	-1
2	3
3	4
4	5
5	-1
6	-1
7	2
8	1
9	0

	intToSeg
0	9
1	8
2	7
3	2
4	3
5	4

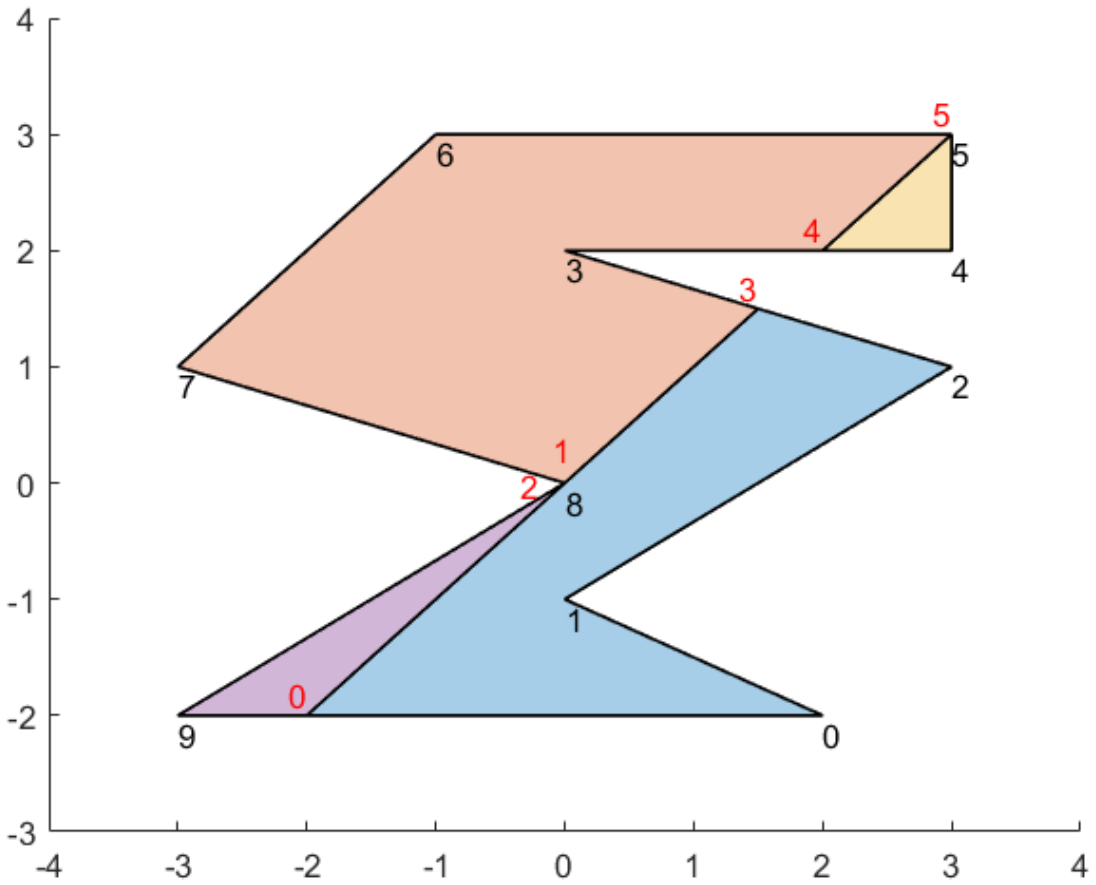
In questo secondo esempio invece i segmenti interni che riguardano il vertice 8 sono 0-1 ed 2-3. In questo caso il vertice non è isolato e l' algoritmo procede con l' unica accortezza di non aggiungere più volte lo stesso punto.



	segToInt
0	-1
1	-1
2	3
3	4
4	5
5	-1
6	-1
7	2
8	1
9	0

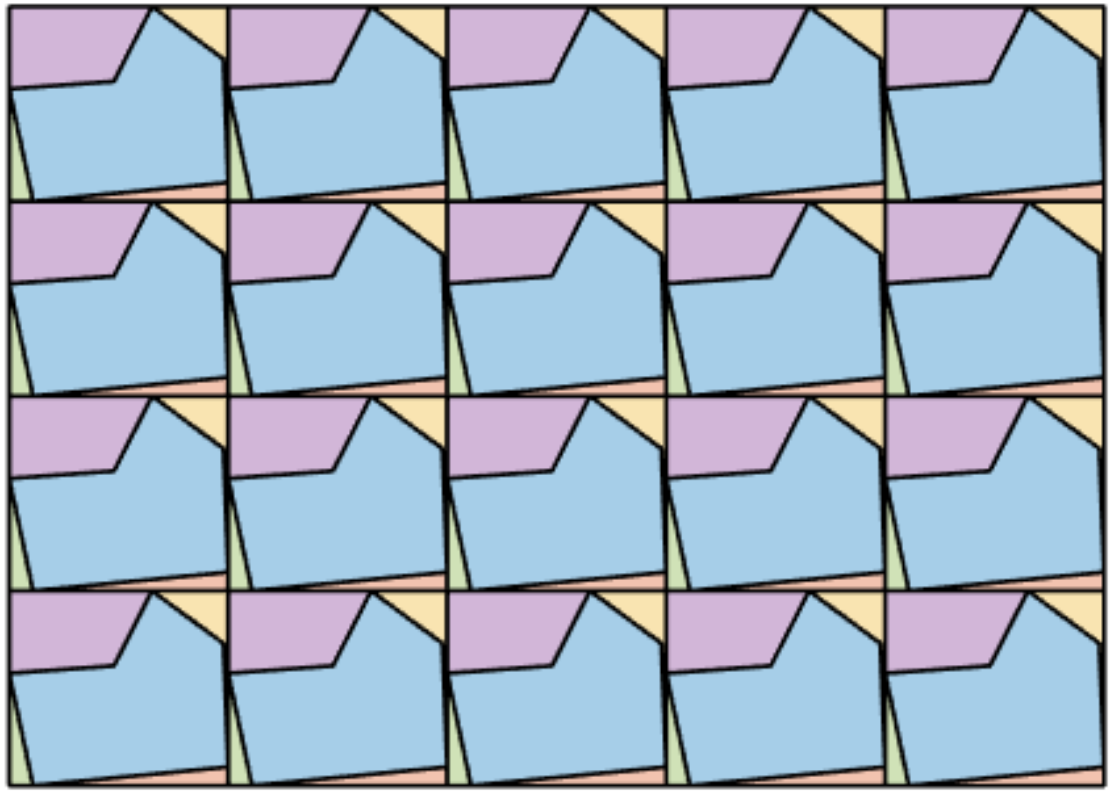
	intToSeg
0	9
1	8
2	7
3	2
4	3
5	4

Il poligono viene suddiviso nei quattro sotto-poligoni.
Notiamo anche che, per le stesse motivazioni dell' esempio del pentagono, è stato aggiunto un solo punto di intersezione in 5.



Progetto 2 e 3

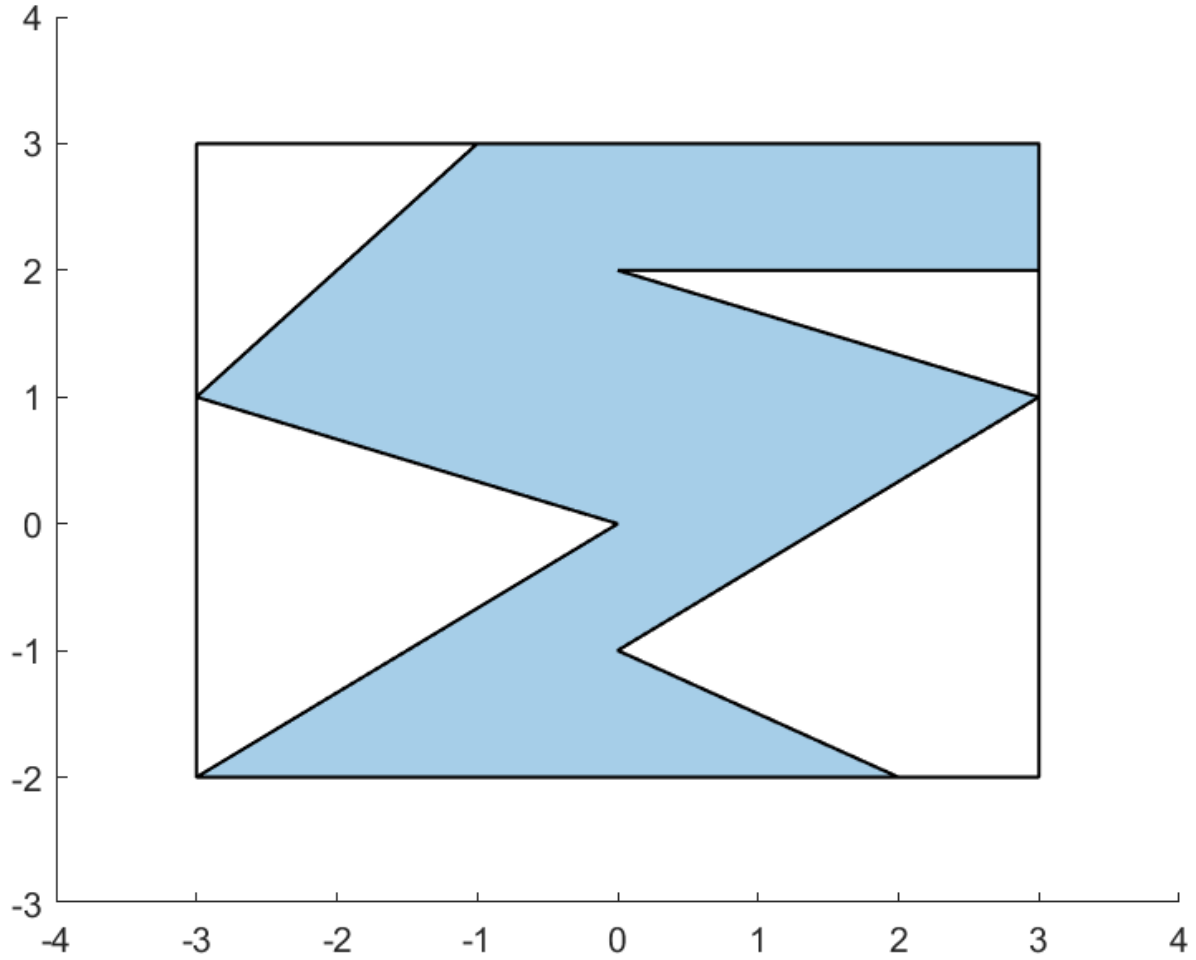
Lo scopo del Progetto 2 è ottenere, a partire da un poligono e da un dominio, nell'immagine rispettivamente un triangolo ed un rettangolo, una mesh, ovvero letteralmente una rete. Bisogna ricoprire quindi con il poligono di partenza l'intero dominio.



Reference Element

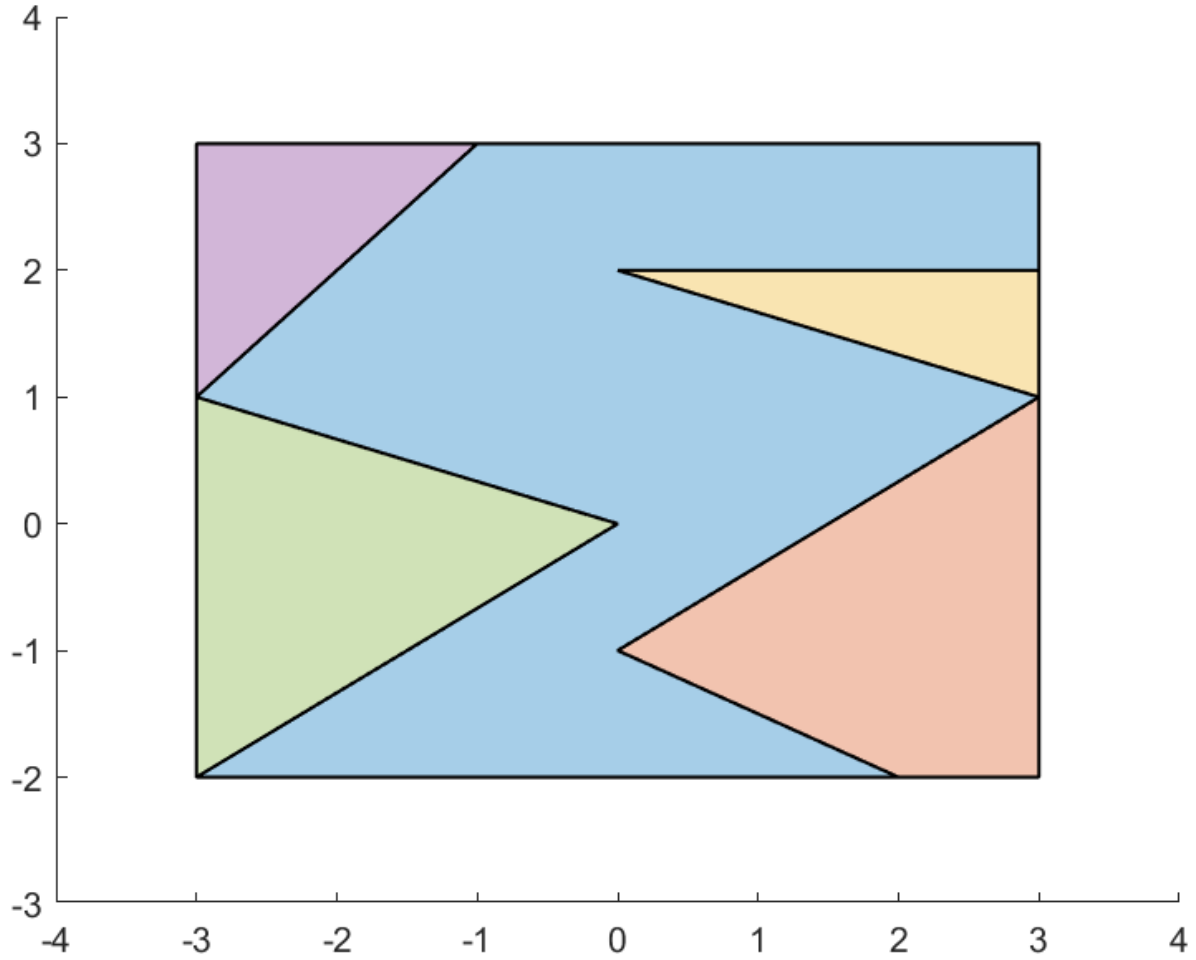
Il primo passo è creare il bounding box del poligono, in modo da poterlo replicare traslando il tutto.

Sono quindi stati trovati il minimo ed il massimo delle coordinate x ed y e generato il rettangolo.



Sono stati trovati tutti I poligoni interni al bounding box, questo permette di ottenere una mesh “piena”.

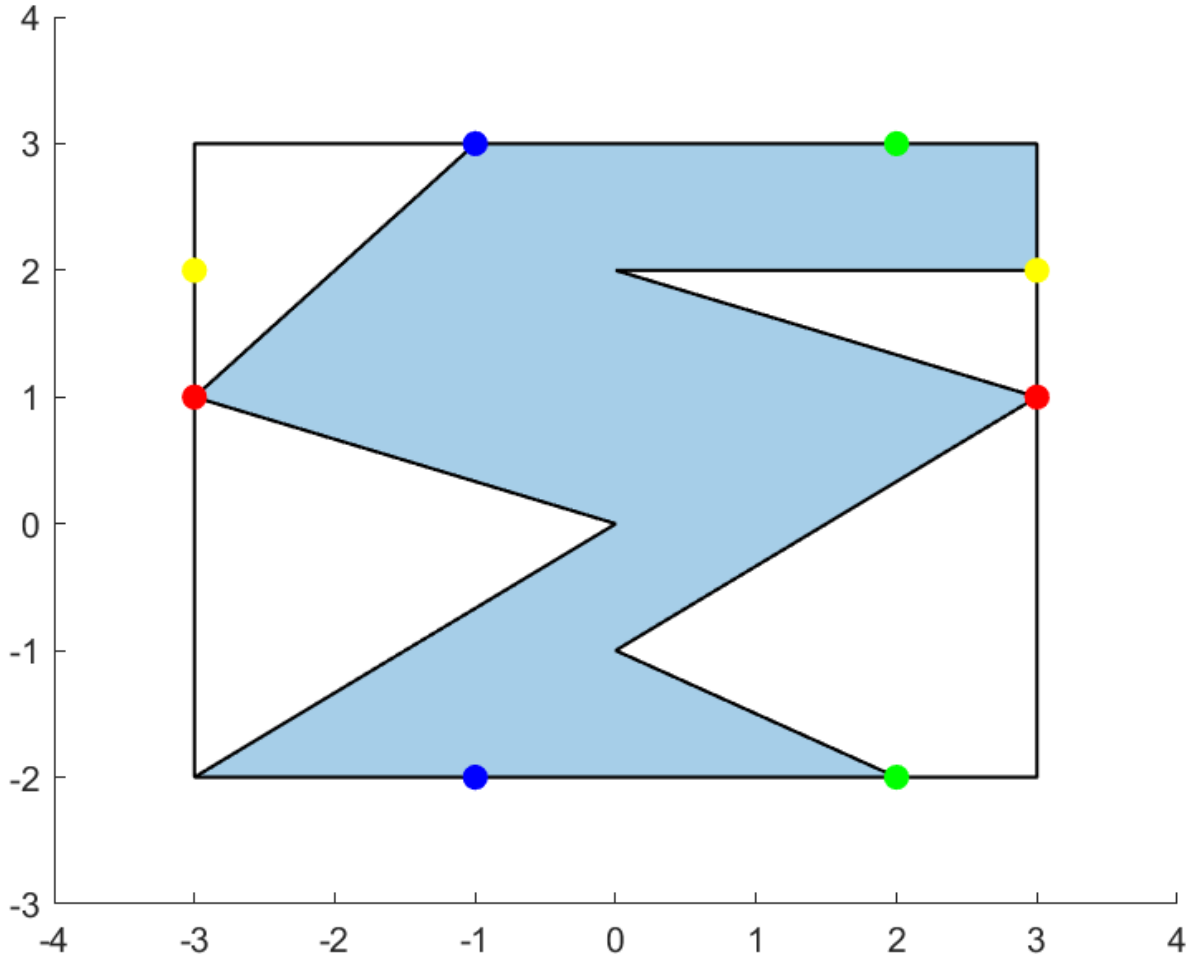
La figura è prodotta dal metodo showReferenceElement che crea uno script MATLAB.



Conforming Element

Si visita ogni lato del poligono per identificare i punti sulla bounding box e riempire le mappe rispettive.

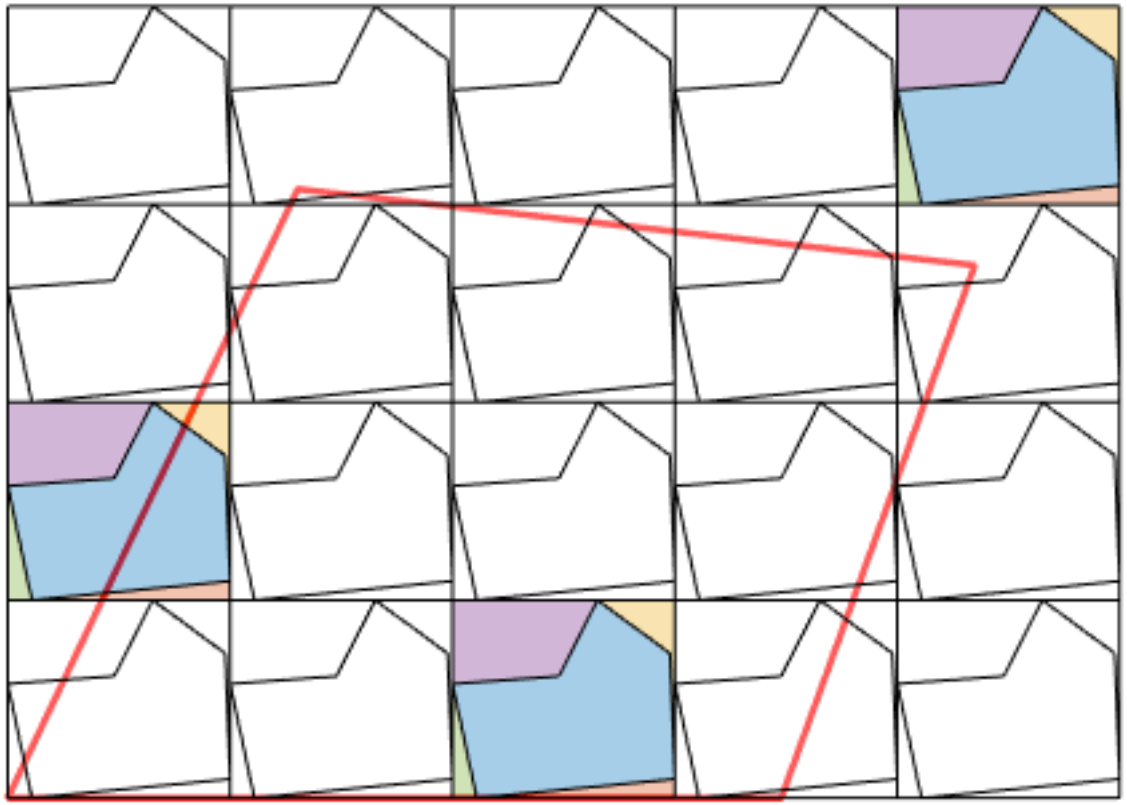
Successivamente li si proietta nel lato opposto della bounding box.



referenceElementInConvexDomain

Dalla definizione di Poligono Convesso:

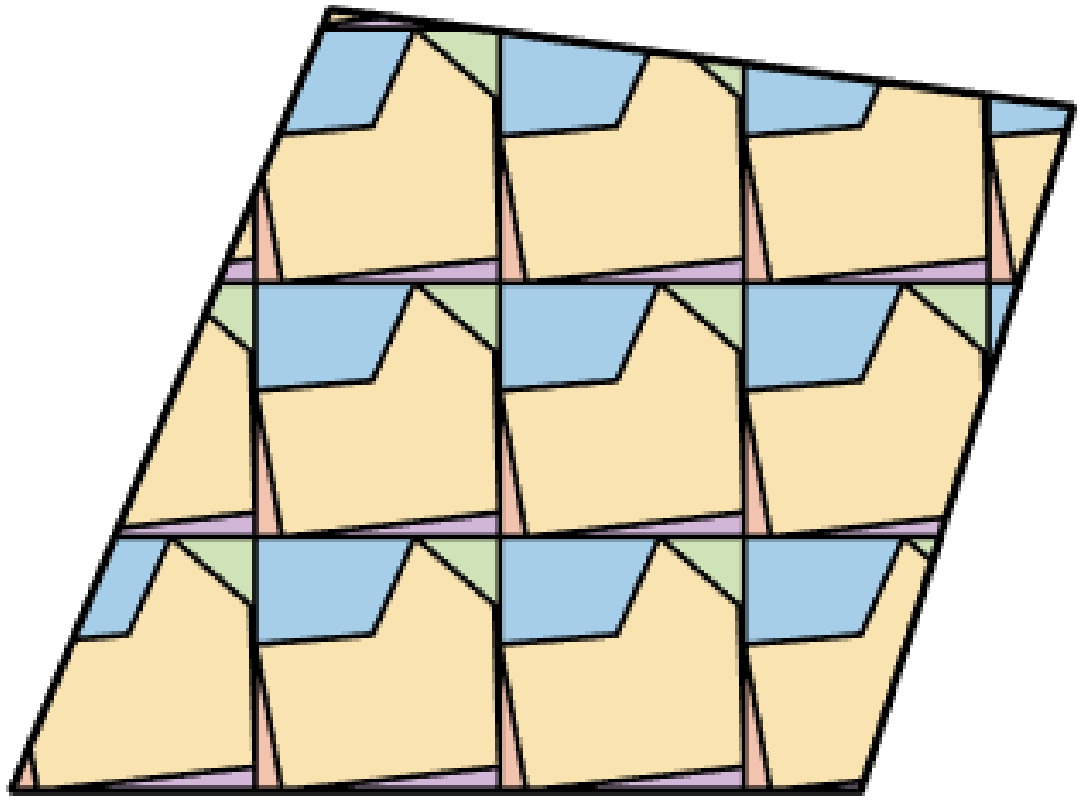
- ReferenceElement interno: I vertici della B.Box sono interni
- ReferenceElement esterno: non è interno e non interseca il dominio



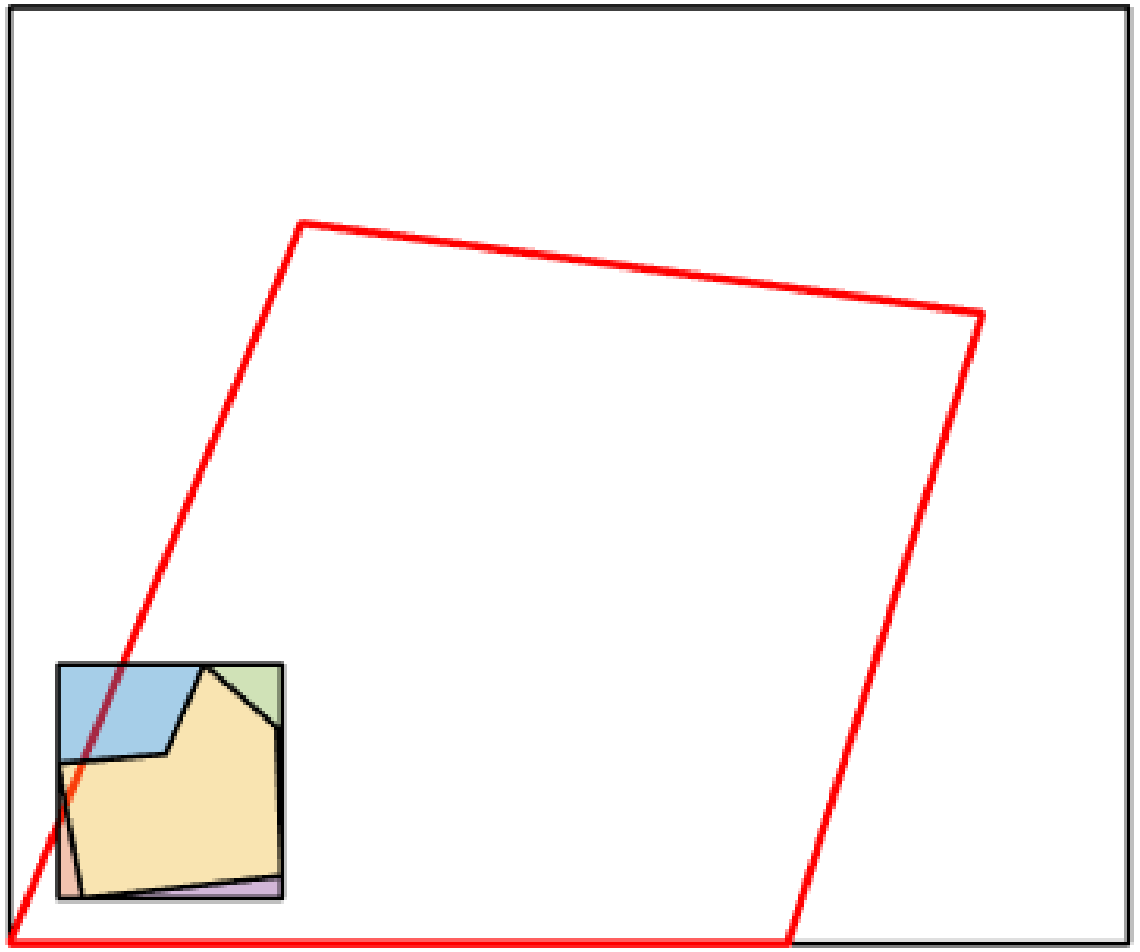
Create Mesh

L algoritmo generale consiste in :

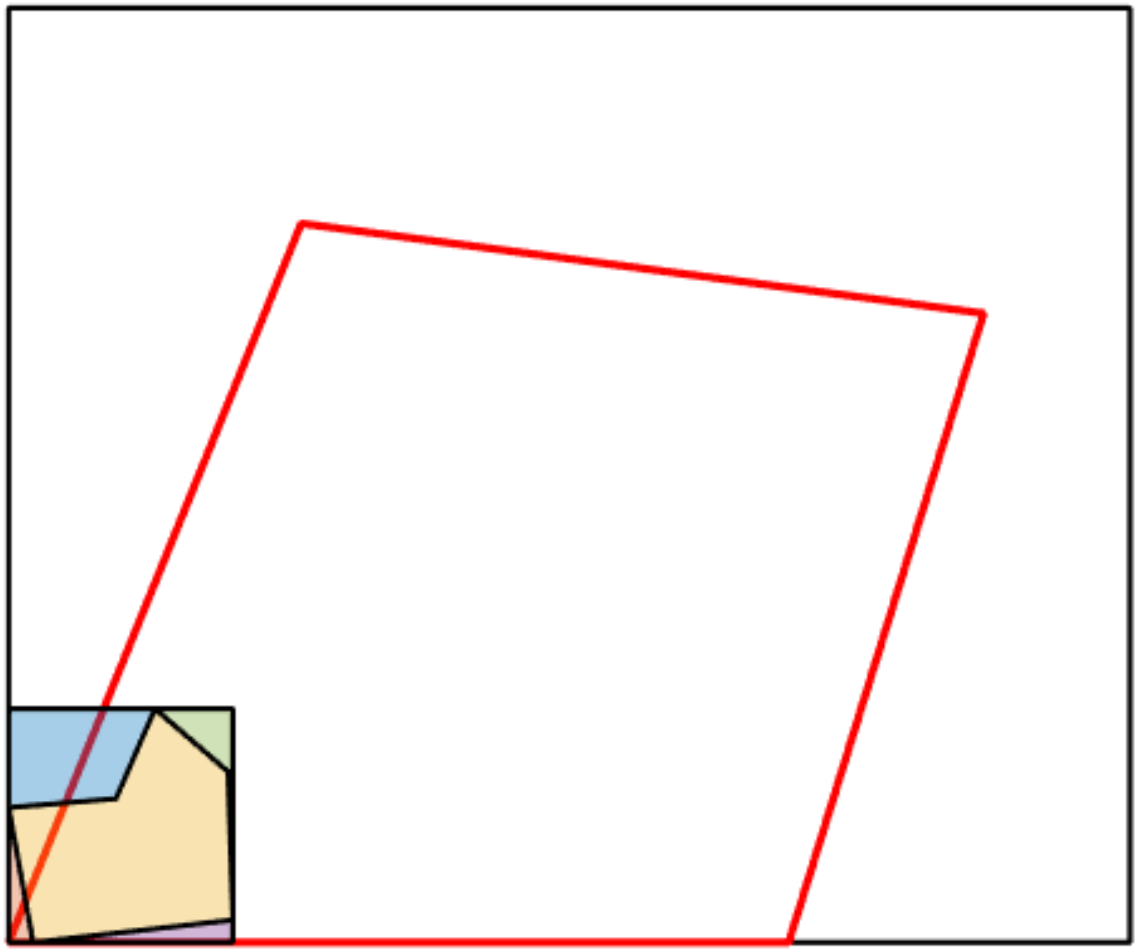
1. impostare il dominio tramite le due funzioni (a seconda della tipologia di dominio) ***setRectangularDomain*** e ***setConvexDomain***



2. Creare il referenceElement del poligono base ed eseguire ***findBoundaryVertices*** and ***MakeConforming*** in modo tale da rendere conforme la mesh.



3. Traslare il ConformingReferenceElement all’origine delle coordinate



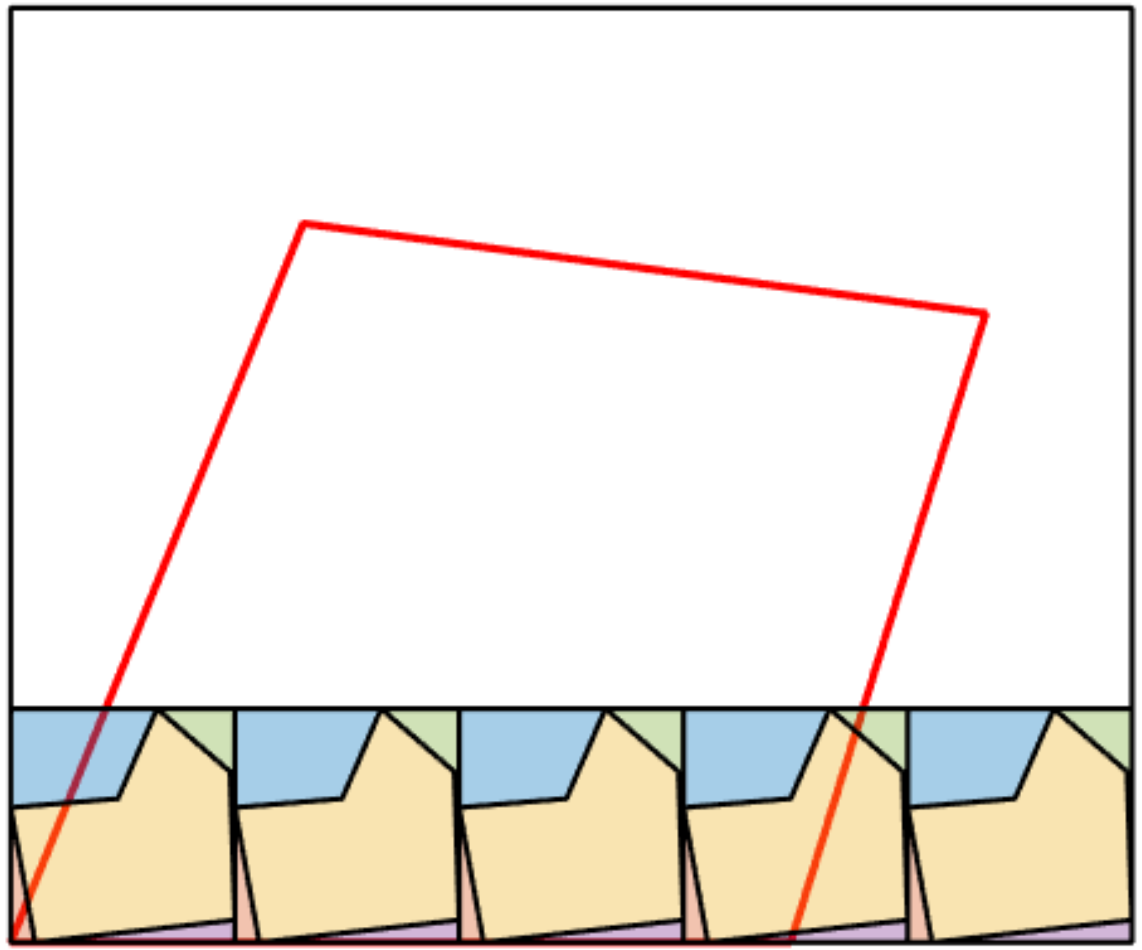
4.

Traslare :
TranslateConformingReferenceElement

Determinare la posizione rispetto al dominio:
referenceElementInConvexDomain

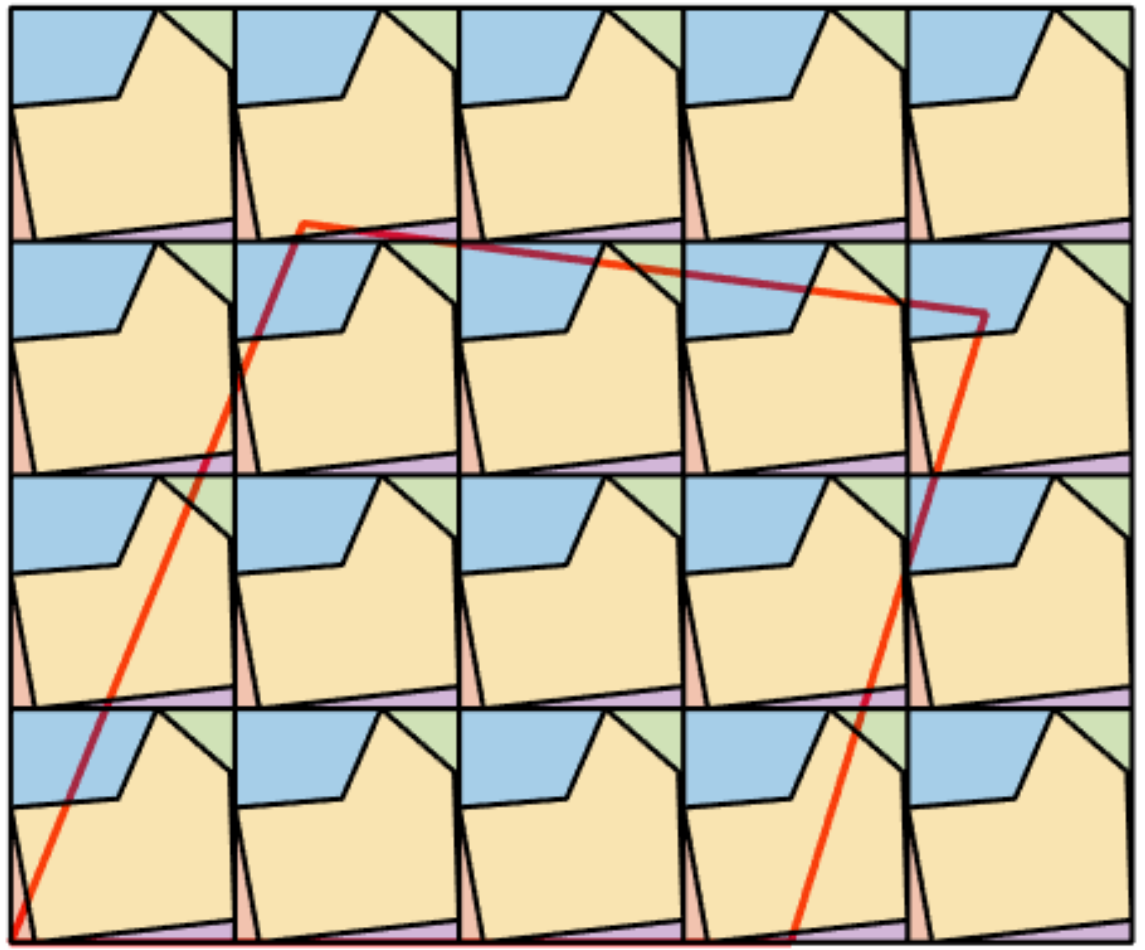
Eventualmente Tagliare e aggiornare translatedReferenceElement:
cutAndDiscardPolygonsOnRightSide

Aggiungere a _meshCells se effettivamente rimane all interno del dominio.

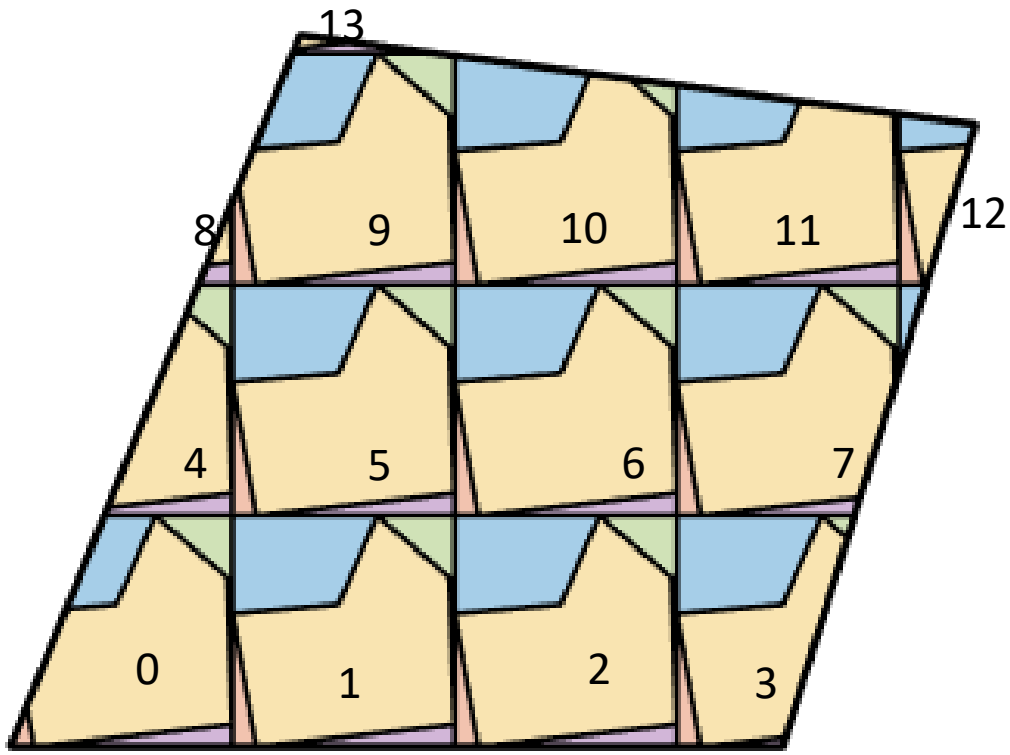


I referenceElement che sono fuori non sono presenti in _meshCells.

I referenceElement che intersecano il dominio verranno tagliato e solamente la parte interna viene aggiunta.



La mesh è composta da 14 celle conformi, ognuna con i poligoni rimasti corrispondenti dopo il taglio



Test con diverse piastrelle e domini