

Progetto Elaborazione di immagini per la chirurgia assistita - A.A. 2022/2023

Prof: Paolo Zaffino

Traccia: Visualizzazione di un campo vettoriale

Studenti:

- Mario Focaccio Matricola #240662
- Erika De Bruno Matricola #243866
- Guido Costanzo Matricola #243461
- Vincenzo Vellone Matricola #237448

SimpleITK Official Documentation



[SimpleITK](#) is an abstraction layer and wrapper around the Insight Segmentation and Registration Toolkit ([ITK](#)). It is available in the following programming languages: C++, Python, R, Java, C#, Lua, Tcl and Ruby.

This notebook demonstrate the use of SimpleITK for interactive image analysis using the Python programming language.

Documentazione per avviare correttamente il presente jupyter notebook

Per eseguire questo notebook, alla data di scrittura, è necessaria la versione di SimpleITK >=2.2.0 o più recente. Questa versione è disponibile presso [PyPi](#) e [Anaconda Cloud](#).

Setup di un ambiente Python

Si consiglia di creare un ambiente virtuale Python separato per eseguire questo notebook di esercitazione. Si consiglia di optare per l'utilizzo della distribuzione Anaconda Python o una semplice distribuzione Python con PIP.

Utilizzando Anaconda

Con [Anaconda](#) è possibile configurare un ambiente virtuale, denominato ad esempio `sitkpy`, e installare tutte le dipendenze, compreso SimpleITK, utilizzando un unico comando:

```
conda env create -f environment.yml
```

Run the notebooks

Per avviare:

```
conda activate sitkpy  
jupyter notebook
```

Per disabilitare l'ambiente virtuale creato:

```
conda deactivate
```

Utilizzando solamente Python

Configurare un ambiente virtuale con python denominato *sitkpy*.

```
python -m venv sitkpy
```

Attivare l'ambiente virtuale:

Su windows: `sitkpy\Scripts\activate.bat`

Su linux/osx: `source sitkpy/bin/activate`

Installare tutti i **pacchetti richiesti** e attivare l'estensione per notebook **ipywidgets**.

```
pip install -r requirements.txt  
jupyter nbextension enable --py --sys-prefix  
widgetsnbextension
```

Il file requirements.txt elenca i pacchetti richiesti ([collegamento qui](#)).

Run the notebooks

Prima di avviarlo, attivare l'ambiente virtuale creato come sopra e nel terminale aprire il notebook jupyter utilizzando il comando:

```
jupyter notebook
```

Obiettivi del progetto

Con l'avvento della chirurgia mini invasiva (MIS) l'approccio della chirurgia è migliorato negli ultimi anni: il medico non agisce più direttamente sul paziente ma esegue scansioni preoperatorie e in base ad esse, pianifica l'intervento.

Le immagini vengono acquisite in fase pre-operatoria e successivamente si effettua la pianificazione: fase in cui si agisce sulle immagini e ciò comporta la segmentazione, il rendering, la modellizzazione e la visualizzazione di superfici.

Questo piano, però, poi deve essere trasferito sull'intra-operatorio;

L'obiettivo è quello di ottenere un'unica immagine data dalla fusione delle due immagini (pre ed intra-operatorio) in modo che esse possano guidare il medico durante l'intervento chirurgico.

Per Processare un'immagine è necessario trovare le corrispondenze geometriche tra le due. Questo può essere realizzato attraverso l'utilizzo di strumenti localizzatori oppure attraverso la registrazione di immagini.

Noi ci occuperemo della registrazione di immagini e, quindi, il nostro obiettivo è quello di trovare delle relazioni per mettere in comunicazione le due immagini e mostrare graficamente lo spostamento dei punti di controllo utilizzati per la registrazione delle immagini sotto descritte.

Durante l'analisi abbiamo lavorato con le sezioni (slice) di una computed tomography (CT) scan. Questo set di dati è costituito da scansioni della testa e del collo precedentemente lavorate e rappresentano un paziente in fase di espirazione ed inspirazione.

Il set di dati, che consiste in due immagini denominate `chest0.nrrd` e `chest5.nrrd` proviene dalle lezioni del corso.

Introduzione: Computed Tomography (CT) Imaging

La tomografia computerizzata (TC) utilizza fasci di raggi X per ottenere differenti intensità di pixel 3D (voxel) del corpo umano. Un catodo riscaldato rilascia fasci ad alta energia (spesso elettroni), che a loro volta rilasciano la loro energia sotto forma di radiazione X. I raggi X attraversano i tessuti del corpo umano e colpiscono un rilevatore sul lato opposto. Un tessuto denso (ad esempio le ossa) assorbe più radiazioni dei tessuti molli (ad esempio il grasso). Quando i raggi X non vengono assorbiti dal corpo (ad esempio nella regione dell'aria all'interno dei polmoni) e raggiungono il rilevatore, vengono visualizzati come neri, simili a una pellicola nera. Al contrario, i tessuti densi sono rappresentati in bianco.

In questo modo, la tomografia computerizzata è in grado di distinguere le differenze di densità e di creare un'immagine 3D del corpo.

Il contenuto effettivo dell'immagine medico\scientifica dipende quindi dallo strumento utilizzato: le fotografie misurano la luce visibile, RX e la TAC misurano l'assorbimento delle radiazioni e gli scanner di MRI misurano l'intensità dei campi magnetici.

Formati di immagine: DICOM e NRRD

NRRD (Near Raw Raster Data) è un formato di file per l'archiviazione e la visualizzazione di dati di immagini mediche. Il suo principale vantaggio rispetto a DICOM, il formato di file standard per le immagini mediche, è che i file NRRD sono anonimizzati e non contengono informazioni sensibili sui pazienti.

Inoltre, i file NRRD possono memorizzare una scansione medica (studio) in un unico file, mentre i set di dati DICOM sono solitamente costituiti da una o più directory che

contengono decine, se non centinaia, di singoli file. L'NRRD è quindi un buon metodo per trasferire i dati delle scansioni mediche per scopi scientifici, proteggendo la privacy dei pazienti.

Strumenti utilizzati

Per condurre questo studio abbiamo utilizzato uno dei più noti linguaggi di programmazione orientato a oggetti che è Python. Ci sono molti IDE Python disponibili su Internet. Uno di questi IDE Python è Anaconda, software open source e uno degli IDE più popolari utilizzati dai programmatore di tutto il mondo. Contiene molti plugin e librerie che ci sono servite per processare e visualizzare le immagini come Matplotlib, SimpleITK. Per costruire le interfacce grafiche abbiamo utilizzato GUI (graphical user interface) un framework che mette a disposizione python.

SimpleITK

SimpleITK è un toolkit per l'analisi delle immagini con un gran numero di componenti che supportano operazioni generali di filtraggio, segmentazione e registrazione delle immagini. Si basa su Insight Segmentation and Registration Toolkit ITK con l'intento di fornire un'interfaccia semplificata a ITK. SimpleITK stesso è scritto in C++ ma è disponibile per un gran numero di linguaggi di programmazione.

Numpy

Con import numpy chiamiamo la libreria NumPy (NUMerical PYthon) che fornisce un nuovo tipo di struttura ai dati chiamata array, che può essere a una o due dimensioni in questo caso avremo matrici.

Questo permette di eseguire in modo efficiente operazioni sui dati sotto forma di array, inoltre, a differenza delle liste, gli array possono contenere solo elementi di un singolo tipo ciò significa che possono essere scritti in memoria in modo più efficiente.

Matplotlib

Matplotlib è una libreria che si utilizza per la creazione di grafici. Viene utilizzato per creare una vasta gamma di visualizzazioni statiche, animate e interattive, come grafici di linea, grafici a dispersione, grafici a barre e istogrammi. Fornisce una vasta gamma di opzioni di personalizzazione, come stili di linea, stili di marcatori e mappe dei colori, che possono essere utilizzate per creare grafici; e funzioni interattive, che possono essere utilizzate per creare visualizzazioni con cui gli utenti possono interagire. La libreria matplotlib.pyplot as plt , dedicata alla rappresentazione grafica, le cui principali funzioni sono: `plt.plot` disegna il grafico in base a un insieme di dati di riferimento.

`plt.xlabel` indica l'unità di misura X sulle ascisse `plt.ylabel` indica l'unità di misura Y sulle ordinate `plt.legend()` indica la posizione e le caratteristiche della legenda `plt.show()` mostra la rappresentazione grafica

Abbiamo anche utilizzato `PyLab` attraverso il comando di sistema `%pylab inline` che è un modulo che appartiene alla libreria Matplotlib. PyLab combina il numpy (modulo numerico) con la grafica pyplot. Gli altri moduli che mette a disposizione Python e che ci

sono serviti per la visualizzazione delle immagini all'interno del notebook sono IPython.display.

In []:

Importazione delle librerie e delle dipendenze necessarie per il notebook

In [1]:

```
%pylab inline
import matplotlib.pyplot as plt

import SimpleITK as sitk
print(sitk.Version())

from myshow import myshow

# Download data to work on (sample ITK)
%run update_path_to_download_script
from downloaddata import fetch_data as fdata

import pylab

%matplotlib notebook
import gui
#import registration_gui as rgui

from IPython import display
# Il codice sotto riportato è obsoleto, aggiornato con quello sopra
from IPython.display import display, HTML

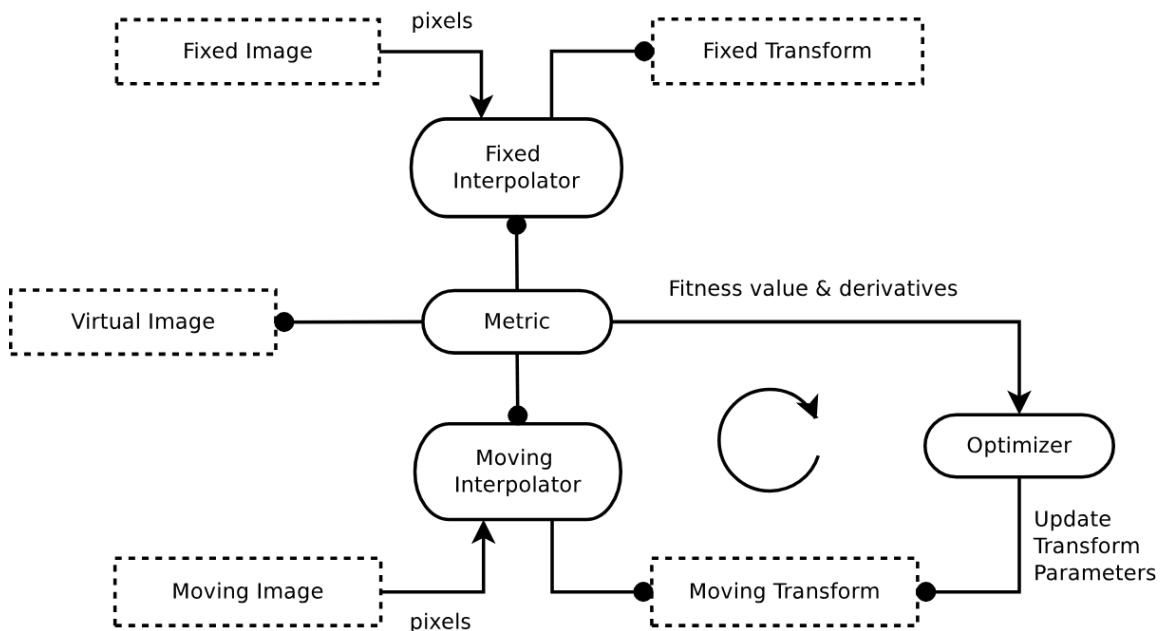
# Per una migliore visualizzazione attivare la grandezza a tutto schermo
display(HTML("<style>.container { width:100% !important; }</style>"))

from __future__ import print_function
```

```
%pylab is deprecated, use %matplotlib inline and import the required libraries.
Populating the interactive namespace from numpy and matplotlib
SimpleITK Version: 2.2.1 (ITK 5.3)
Compiled: Dec 7 2022 19:23:57
```

Definizione di fixed e moving

Una volta definiti gli strumenti che ci servono, impostiamo il lavoro seguendo questo schema illustrato. L'obiettivo dello studio è quello di dare due etichette: fixed e moving, queste vanno attribuite, una per l'immagine pre-operatorie e una per l'immagine intra operatoria al fine di ottenere una fusione delle due e quindi, acquisire l'immagine registrata.

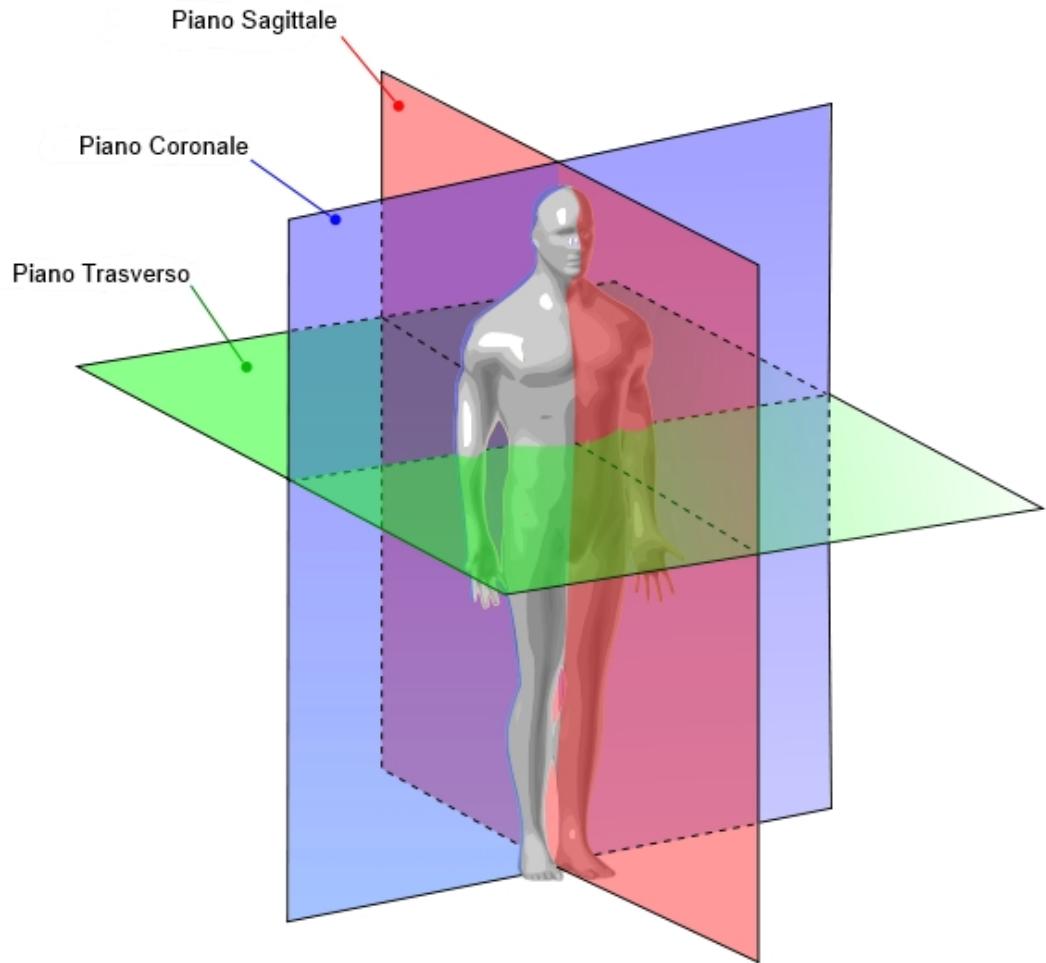


```
In [2]: # Importo i file dalla cartella in locale ./chest4D
fixed_image = sitk.ReadImage("./chest4D/chest0.nrrd", sitk.sitkFloat32)
fixed_np = sitk.GetArrayFromImage(fixed_image)

moving_image = sitk.ReadImage("./chest4D/chest5.nrrd", sitk.sitkFloat32)
moving_np = sitk.GetArrayFromImage(moving_image)
```

Esplorazione del dataset (2 volumi .nrrd)

Method name	Accessed *.mhd parameter
GetDimension()	NDims
GetSize()	DimSize
GetOrigin()	CenterOfRotation
GetSpacing()	ElementSpacing
GetDirection()	TransformMatrix



```
In [3]: # Leggo le informazioni delle immagini
```

```
print (fixed_image.GetDimension(), moving_image.GetDimension())
3 3
```

```
In [4]: # Richiedo le caratteristiche delle immagini importate
```

```
print (fixed_image.GetSize())
print (fixed_image.GetOrigin())
print (fixed_image.GetSpacing())
print (fixed_image.GetDirection())
print (fixed_image.GetNumberOfComponentsPerPixel())

(512, 512, 136)
(-242.5, -242.5, -197.5)
(0.9472659826278687, 0.9472659826278687, 2.5)
(1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0)
1
```

```
In [5]: # Visualizzo le immagini nel piano trasverso
```

```
#fixed_window_level = [932, 60]
#moving_window_level = [932, 95]

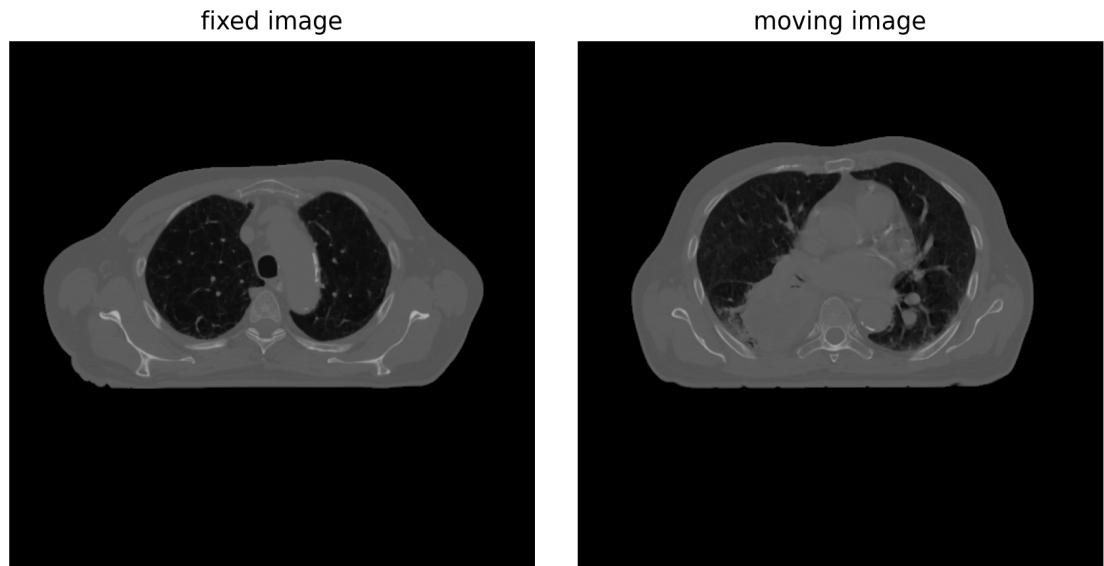
gui.MultiImageDisplay(
    image_list=[fixed_image, moving_image],
    title_list=["fixed image", "moving image"],
    figure_size=(8, 4),
    #window_level_list=[fixed_window_level, moving_window_level],
```

```

        intensity_slider_range_percentile=[0, 100],
    );

```

VBox(children=(Box(children=(IntSlider(value=67, description='image slice:', max=135), IntSlider(value=67, des...



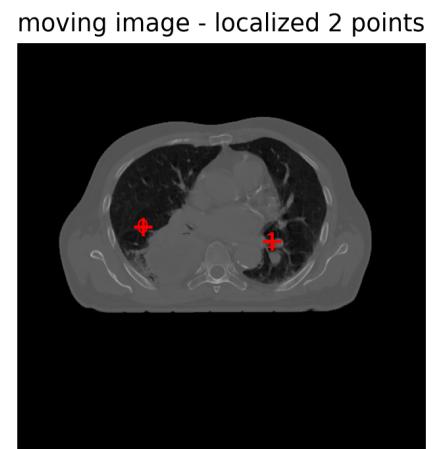
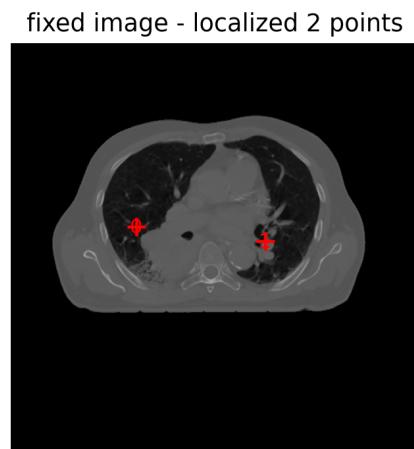
```

In [6]: initial_transform = sitk.CenteredTransformInitializer(
    fixed_image,
    moving_image,
    sitk.Euler3DTransform(),
    sitk.CenteredTransformInitializerFilter.GEOMETRY,
)

gui.RegistrationPointDataAquisition(
    fixed_image,
    moving_image,
    figure_size=(8, 4),
    known_transformation=initial_transform,
    #fixed_window_level=fixed_window_level,
    #moving_window_level=moving_window_level,
);

```

HBox(children=(HBox(children=(Box(children=(RadioButtons(description='Interaction mode:', options=('edit', 'vi...



Normalizzazione delle intensità delle immagini

La normalizzazione di due immagini cliniche (nel caso specifico si tratta di immagini in formato **NRD** (*Nrrd-ItkImage*)) è un processo essenziale per garantire la corretta comparabilità e l'analisi accurata delle immagini. Inizialmente, è importante capire che la normalizzazione si riferisce a una serie di operazioni volte a rendere uniformi le intensità dei pixel nelle immagini, in modo che siano confrontabili tra loro e con uno standard di riferimento.

La normalizzazione delle immagini cliniche è cruciale per vari motivi. Innanzitutto, consente di eliminare le variazioni sistematiche nell'intensità dei pixel che potrebbero essere causate da fattori esterni come le condizioni di acquisizione o le differenze nei parametri di imaging. Queste variazioni possono influenzare negativamente l'accuratezza dei metodi di analisi e rendere difficile il confronto tra diverse immagini o pazienti.

Perché è importante effettuare una normalizzazione delle immagini in ambito clinico prima di effettuare una registrazione delle immagini?

La FFD è una tecnica di registrazione che consente di allineare due immagini utilizzando una griglia deformabile. Tuttavia, la registrazione **FFD** mediante **Spline** o **B-Spline** può essere influenzata da differenze significative nelle intensità dei pixel tra le immagini di input. La normalizzazione delle immagini riduce queste differenze, migliorando la corrispondenza tra le strutture anatomiche nelle immagini e consentendo una registrazione più accurata.

```
In [7]: # Estraggo le proprietà delle immagini
min_intensity = np.min([np.min(fixed_np), np.min(moving_np)])
max_intensity = np.max([np.max(moving_np), np.max(moving_np)])
print("Min:", min_intensity, "Max:", max_intensity)

Min; -1200.0 Max: 3071.0
```

```
In [8]: print(np.max(moving_np), np.max(moving_np))

3071.0 3071.0
```

```
In [9]: print(np.min(fixed_np), np.min(moving_np))

-1200.0 -1200.0
```

Normalizzazione dell'intensità dell'immagine in un range [-1, 1]

```
In [10]: # Normalise the image to fit [-1, 1] range:
normalized_fixed = sitk.Normalize(fixed_image)
normalized_moving = sitk.Normalize(moving_image)

fixed_npNorm = sitk.GetArrayFromImage(normalized_fixed)
moving_npNorm = sitk.GetArrayFromImage(normalized_moving)

min_intensity = np.min([np.min(fixed_npNorm), np.min(moving_npNorm)])
max_intensity = np.max([np.max(fixed_npNorm), np.max(moving_npNorm)])
print("Min:", min_intensity, "Max:", max_intensity)

Min; -0.5953140487025191 Max: 7.534555287076418
```

```
In [11]: print(np.min(fixed_npNorm), np.min(moving_npNorm))

-0.5916272341037799 -0.5953140487025191
```

```
In [12]: print(np.max(fixed_npNorm), np.max(moving_npNorm))  
7.534555287076418 7.453507452899429
```

```
In [ ]:
```

Nel codice sopra, si utilizza la funzione `GetArrayFromImage` per ottenere un array numpy dalle immagini di input. Successivamente, si calcola il valore minimo e massimo di intensità tra le due immagini utilizzando le funzioni `np.min` e `np.max`.

Infine, si applica la normalizzazione utilizzando la funzione `sitk.Normalize`, che ridimensiona l'intervallo di intensità delle immagini.

Questa operazione non è fondamentale nel nostro caso perché come si evidenzia nelle righe sopra indicate, i valori delle due immagini di partenza risultavano normalizzati entro gli stessi intervalli di minimo e massimo.

Metodi di registrazione di un'immagine clinica

La pratica clinica si basa prevalentemente su due tipologie di registrazione che si identificano in:

- **Manuale:** significa visualizzare i due dataset e spostare manualmente l'oggetto in modo tale che questo corrisponda a dei siti anatomici di interesse. È una trasformazione rigida, la cui accuratezza dipende dall'esperienza e dal giudizio dell'operatore. Si basa sull'utilizzo di dati DICOM: ogni scanner ha un sistema di riferimento solidale al paziente, si possono allineare i due dataset diversi allineandone i centri
- **Automatica:** si utilizzano le matrici per capire se le due immagini sono allineate o no, e si utilizzano due tecniche:
 - **Intensity-based:** si calcola l'operazione di trasformazione confrontando iterazione per iterazione i livelli di grigio dei due dataset di immagini. Il confronto avviene sull'intensità dei livelli di grigio delle immagini e si basa sulla minimizzazione di una funzione di disparità
 - **Geometry-based:** si basa sull'identificazione di punti fiduciali artificiali (marcatori, clips) o su landmark anatomici segmentati (punti, profili, etc). Si stabilisce che questi punti devono essere allineati. Per esempio: marker radio-opachi che vengono dati al paziente

Il metodo che abbiamo sfruttato noi per questo studio si basa sulla registrazione automatica per intensity-based. Questo metodo ci ha permesso di trarre profitto dalla sua tecnica andando a dividere l'immagine con una griglia in modo da identificare le zone di interesse che vogliamo e sovrapporre con esattezza.

La griglia può essere risolta tanto quanto l'immagine (se ho un'immagine 512x512 avrò una griglia 512x512 e dovrò calcolare la trasformazione per ogni voxel dell'immagine perché la griglia e l'immagine corrispondono). La definizione della griglia definisce il dettaglio dell'immagine che si considera nell'ottimizzazione della funzione di costo.

Un'immagine che ha delle coordinate chiamate pixel. In un punto dell'immagine io so di essere al pixel (x,y) o nel voxel (x,y,z). La trasformazione che si calcola nello spazio reale, perché devo sapere se il paziente, lo strumento o il lettino può essere spostato di una quantità definibile in mm, non in pixel. Allora quando si calcola la trasformazione la prima operazione che si fa è quella di trasformare le coordinate immagine in coordinate fisiche, poi si calcola la trasformazione in coordinate fisiche (definire ad esempio una rototraslazione in coordinate fisiche) e, per spostare l'immagine nel momento in cui si applica, si riportano le coordinate fisiche a coordinate immagine.

Trasformazione rigida e non rigida

Il processo di registrazione è un processo iterativo che si ferma nel momento in cui raggiunge un certo obiettivo per la funzione di costo. Quali sono gli ingredienti da mettere dentro l'algoritmo per stabilire l'iterazione e le condizioni per uscirne?

Innanzitutto, si definisce l'immagine fixed (resterà ferma), di riferimento, che deve andare ad accogliere l'immagine che si muove (moving image). Per la moving image si dovrà calcolare quella trasformazione che porterà la moving sulla fixed. Quindi, definita B l'immagine moving e definita A l'immagine fixed, si deve calcolare la trasformazione che porta B su A. Nel caso di operazione lineare si parla di registrazione rigida cioè al massimo si può traslare, ruotare o rototraslare una delle due immagini che si stanno considerando (chiaramente la fixed viene traslata sulla moving). Le distanze relative tra i punti dell'immagine non cambiano. A seconda che si trasli, ruoti o rototrasli un'immagine sull'altra abbiamo rispettivamente 1, 3 o 6 gradi di libertà. Per quanto riguarda le trasformazioni non rigide possiamo avere la registrazione di tipo affine e proiettiva dove si effettuano più operazioni di scalatura arrivando anche a 12 gradi di libertà e la deformabile dove è possibile modificare le distanze in maniera non lineare una rispetto all'altra.

Non-Rigid Registration: Free Form Deformation

Nell'imaging clinico, la registrazione non rigida rappresenta una tecnica fondamentale per l'allineamento accurato delle immagini mediche. La registrazione non rigida permette di adattare morfologicamente un'immagine di riferimento a un'immagine target, considerando anche le deformazioni locali che possono verificarsi a causa di movimenti o patologie. Tra i vari algoritmi di registrazione non rigida, il metodo di Free Form Deformation (FFD) ha dimostrato di essere particolarmente efficace nell'affrontare queste sfide.

Definizione di Free Form Deformation:

Il Free Form Deformation è un algoritmo di registrazione non rigida ampiamente utilizzato nell'ambito dell'imaging clinico. Esso si basa sulla concettualizzazione dell'immagine come una griglia di punti di controllo sparsi uniformemente nello spazio. Questi punti di controllo vengono posizionati sull'immagine di riferimento e vengono successivamente deformate per adattare l'immagine di riferimento all'immagine target.

Il processo di Free Form Deformation:

Il processo di Free Form Deformation comprende diversi passaggi chiave. Inizialmente, una griglia regolare di punti di controllo viene posizionata sull'immagine di riferimento. Questi punti di controllo definiscono la struttura della deformazione. Successivamente, i punti di controllo vengono spostati in modo da adattare la forma dell'immagine di riferimento all'immagine target. La deformazione dei punti di controllo avviene utilizzando funzioni matematiche, come ad esempio spline o B-spline, che consentono una flessibilità nella modellazione delle deformazioni locali.

Vantaggi della registrazione non rigida con Free Form Deformation:

L'utilizzo di Free Form Deformation nella registrazione non rigida offre diversi vantaggi nell'ambito dell'imaging clinico. Prima di tutto, permette di catturare le deformazioni locali che possono essere trascurate dai metodi di registrazione rigida. Questo è particolarmente utile quando si lavora con immagini mediche che presentano deformazioni non lineari, ad esempio a causa di movimenti degli organi o di malattie che alterano la forma degli stessi. Inoltre, il FFD può essere utilizzato in combinazione con altre tecniche di registrazione, consentendo di ottenere risultati più accurati e completi.

Applicazioni dell'algoritmo di Free Form Deformation:

L'algoritmo di Free Form Deformation ha numerose applicazioni nell'ambito dell'imaging clinico. Una delle sue principali applicazioni è nella guida interventistica, dove la registrazione non rigida consente di tracciare i cambiamenti anatomici in tempo reale durante un intervento. Inoltre, il FFD trova impiego anche nell'analisi di serie temporali di immagini, come ad esempio l'analisi delle variazioni di forma degli organi nel tempo.

L'utilizzo del Free Form Deformation (FFD) in combinazione con altre tecniche di registrazione può portare a risultati più accurati e completi nell'imaging medico. L'integrazione del FFD con la registrazione affine, basata su intensità, basata su caratteristiche anatomiche e multi-modale consente di catturare e modellare le deformazioni non lineari, le variazioni di intensità e le differenze tra modalità di acquisizione, portando a un allineamento più preciso e robusto delle immagini cliniche. Queste combinazioni di tecniche offrono un potenziale significativo per l'imaging e la navigazione intra-operatoria.

Tecniche di registrazione con cui la Free Form Deformation può essere combinata:

- 1. Registrazione affine:** La registrazione affine rappresenta una tecnica di registrazione rigida estesa che permette traslazioni, rotazioni, scalature e deformazioni lineari. Integrare il FFD con la registrazione affine consente di catturare meglio le deformazioni non lineari presenti nelle immagini mediche. La registrazione affine viene solitamente applicata come prima fase, allineando le strutture principali, seguita dal FFD per modellare le deformazioni locali.
- 2. Registrazione basata su intensità:** La registrazione basata su intensità si basa sulla comparazione delle caratteristiche di intensità delle immagini. Questa tecnica è utile quando le deformazioni sono associate a variazioni di intensità locali o globali. Utilizzando il FFD in combinazione con la registrazione basata su intensità, è

possibile catturare e modellare in modo accurato le variazioni di intensità all'interno delle immagini, consentendo una registrazione più precisa.

3. Registrazione basata su caratteristiche anatomiche: La registrazione basata su caratteristiche anatomiche coinvolge l'individuazione e l'allineamento di punti di riferimento anatomici comuni tra le immagini. Questi punti di riferimento possono essere strutture anatomiche, come ad esempio vertici di superfici o centri di massa di organi. Integrare il FFD con la registrazione basata su caratteristiche anatomiche permette di catturare e modellare le deformazioni locali associate a specifiche strutture, migliorando la precisione dell'allineamento.

4. Registrazione multi-modale: La registrazione multi-modale coinvolge l'allineamento di immagini provenienti da diverse modalità di acquisizione, ad esempio immagini a raggi X e immagini MRI. L'integrazione del FFD con la registrazione multi-modale permette di considerare e modellare le differenze di contrasto e intensità presenti tra le immagini, migliorando la corrispondenza tra le strutture anatomiche e consentendo una registrazione più accurata e robusta.

Implementazione FFD utilizzando B-Spline

A deformable transform over a bounded spatial domain using a BSpline representation for a 2D or 3D coordinate space.

SimpleITK Doc

(<https://simpleitk.readthedocs.io/en/v1.2.0/Examples/ImageRegistrationMethodBSpline.html>)

Cenni teorici alla B-Spline

Una **BSplineTransform** ha di solito un gran numero di parametri che aumentano la complessità e la durata dell'ottimizzazione della deformazione. (*L'esempio sotto riportato ha durata di 1h e 30min su MacBook Pro M1 2020*). L'approccio BSpline multi-risoluzione esegue inizialmente la registrazione a una risoluzione inferiore con meno parametri al primo livello e poi adatta o ricampiona i punti di controllo BSpline a una risoluzione superiore ai livelli successivi. L'adattamento della trasformazione avviene in concomitanza con la funzione multilivello di *ImageRegistrationMethod*. Ciò comporta inevitabilmente una variazione per ogni livello di risoluzione dei seguenti parametri:

- Sigma di smoothing
- Percentuale di campionamento
- Risoluzione della BSpline Al fine di risolvere in modo efficiente una serie di problemi di registrazione.

I modelli basati su Bsplines, vanno a costruire la griglia sull'immagine definendo sull'immagine dei nodi e, all'intersezione dei nodi vengono messi dei pesi; Le coordinate e le dimensioni della griglia (espressa in mm) la sceglio noi in base alla deformazione che vogliamo ottenere. Sui nodi Bsplines deforme l'immagine per fare in modo che i livelli di grigio della fixed corrispondano ai livelli di grigio della moving. Trovo così un campo vettoriale che mi fa diventare simili le due immagini, ogni vettore rappresenterà lo spostamento e la deformazione dell'immagine per ogni nodo della griglia. Il problema

della griglia è che non descrive bene l'anatomia del paziente, poiché è vero che vogliamo ottenere un'unica immagine del paziente, ma questa deve essere coerente con la sua anatomia per essere una buona immagine diagnostica. Questo si può risolvere con la regolarizzazione a zone con slicer 3D.

Output ottenuto alla fine della registrazione con Bsplines

Alla fine della registrazione e dell'esecuzione di tutti gli script presenti in questo paragrafo del notebook, avremo due file di output:

```
registrazione_outTx.tfm  
output_bspline.nii.gz
```

La prima rappresenta la trasformazione della registrazione B-Spline, mentre la seconda consiste nell'intero volume registrato mediante free form deformation.

```
In [13]: def command_iteration(method, bspline_transform) :  
    if method.GetOptimizerIteration() == 0:  
        # The BSpline is resized before the first optimizer  
        # iteration is completed per level. Print the transform object  
        # to show the adapted BSpline transform.  
        print(bspline_transform)  
  
    print("{0:3} = {1:10.5f}".format(method.GetOptimizerIteration(),  
                                    method.GetMetricValue()))  
  
  
def command_multi_iteration(method) :  
    # The sitkMultiResolutionIterationEvent occurs before the  
    # resolution of the transform. This event is used here to print  
    # the status of the optimizer from the previous registration level.  
    if R.GetCurrentLevel() > 0:  
        print("Optimizer stop condition: {0}".format(R.GetOptimizerStopConditionType()))  
        print(" Iteration: {0}".format(R.GetOptimizerIteration()))  
        print(" Metric value: {0}".format(R.GetMetricValue()))  
  
    print("----- Resolution Changing -----")  
  
    transformDomainMeshSize=[2]*fixed_image.GetDimension()  
    tx = sitk.BSplineTransformInitializer(fixed_image,  
                                         transformDomainMeshSize )  
  
    print("Initial Number of Parameters: {0}".format(tx.GetNumberOfParameters()))  
  
    R = sitk.ImageRegistrationMethod()  
    R.SetMetricAsJointHistogramMutualInformation()  
  
    R.SetOptimizerAsGradientDescentLineSearch(5.0,  
                                              100,  
                                              convergenceMinimumValue=1e-4,  
                                              convergenceWindowSize=5)  
  
    R.SetInterpolator(sitk.sitkLinear)  
    R.SetInitialTransformAsBSpline(tx,
```

```

        inPlace=True,
        scaleFactors=[1,2,5])
R.SetShrinkFactorsPerLevel([4,2,1])
R.SetSmoothingSigmasPerLevel([4,2,1])

R.AddCommand( sitk.sitkIterationEvent, lambda: command_iteration(R, tx) )
R.AddCommand( sitk.sitkMultiResolutionIterationEvent, lambda: command_multi_
outTx = R.Execute(fixed_image, moving_image)

print("-----")
print(tx)
print(outTx)
print("Optimizer stop condition: {0}".format(R.GetOptimizerStopConditionDesc
print(" Iteration: {0}".format(R.GetOptimizerIteration())))
print(" Metric value: {0}".format(R.GetMetricValue()))

sitk.WriteTransform(outTx, 'registrazione_outTx.tfm')

if ( not "SITK_NOSHOW" in os.environ ):

    resampler = sitk.ResampleImageFilter()
    resampler.SetReferenceImage(fixed_image);
    resampler.SetInterpolator(sitk.sitkLinear)
    resampler.SetDefaultPixelValue(100)
    resampler.SetTransform(outTx)

    out = resampler.Execute(moving_image)
    simg1 = sitk.Cast(sitk.RescaleIntensity(fixed_image), sitk.sitkUInt8)
    simg2 = sitk.Cast(sitk.RescaleIntensity(out), sitk.sitkUInt8)
    cimg = sitk.Compose(simg1, simg2, simg1//2.+simg2//2.)
    sitk.Show( cimg, "Image Registration Composition" )

```

```

Initial Number of Parameters: 375
----- Resolution Changing -----
itk::simple::BSplineTransform
BSplineTransform (0x11ae0a2d0)
    RTTI typeinfo: itk::BSplineTransform<double, 3u, 3u>
    Reference Count: 9
    Modified Time: 7695
    Debug: Off
    Object Name:
    Observers:
        none
    CoefficientImage: [ 0x11b7ee1f0, 0x11b7f58c0, 0x11b7f4660 ]
    TransformDomainOrigin: [-241.079, -241.079, -193.75]
    TransformDomainPhysicalDimensions: [484.053, 484.053, 337.5]
    TransformDomainDirection: 1 0 0
0 1 0
0 0 1

    TransformDomainMeshSize: [2, 2, 2]
    GridSize: [5, 5, 5]
    GridOrigin: [-483.106, -483.106, -362.5]
    GridSpacing: [242.026, 242.026, 168.75]
    GridDirection: 1 0 0
0 1 0
0 0 1

0 = -0.89897
1 = -0.89880
2 = -0.89874
3 = -0.89874
Optimizer stop condition: GradientDescentLineSearchOptimizerv4Template: Conv
ergence checker passed at iteration 4.
Iteration: 4
Metric value: -0.8989726219744086
----- Resolution Changing -----
itk::simple::BSplineTransform
BSplineTransform (0x11ae0a2d0)
    RTTI typeinfo: itk::BSplineTransform<double, 3u, 3u>
    Reference Count: 9
    Modified Time: 24727
    Debug: Off
    Object Name:
    Observers:
        none
    CoefficientImage: [ 0x11b7ee1f0, 0x11b7f58c0, 0x11b7f4660 ]
    TransformDomainOrigin: [-242.026, -242.026, -196.25]
    TransformDomainPhysicalDimensions: [484.053, 484.053, 337.5]
    TransformDomainDirection: 1 0 0
0 1 0
0 0 1

    TransformDomainMeshSize: [4, 4, 4]
    GridSize: [7, 7, 7]
    GridOrigin: [-363.04, -363.04, -280.625]
    GridSpacing: [121.013, 121.013, 84.375]
    GridDirection: 1 0 0
0 1 0
0 0 1

0 = -1.04912
1 = -1.04913
2 = -1.04914

```

```

3 = -1.04922
Optimizer stop condition: GradientDescentLineSearchOptimizerv4Template: Conv
ergence checker passed at iteration 4.
Iteration: 4
Metric value: -1.0492162959465912
----- Resolution Changing -----
itk::simple::BSplineTransform
BSplineTransform (0x11ae0a2d0)
RTTI typeinfo: itk::BSplineTransform<double, 3u, 3u>
Reference Count: 9
Modified Time: 39421
Debug: Off
Object Name:
Observers:
none
CoefficientImage: [ 0x11b7ee1f0, 0x11b7f58c0, 0x11b7f4660 ]
TransformDomainOrigin: [-242.5, -242.5, -197.5]
TransformDomainPhysicalDimensions: [484.053, 484.053, 337.5]
TransformDomainDirection: 1 0 0
0 1 0
0 0 1

TransformDomainMeshSize: [10, 10, 10]
GridSize: [13, 13, 13]
GridOrigin: [-290.905, -290.905, -231.25]
GridSpacing: [48.4053, 48.4053, 33.75]
GridDirection: 1 0 0
0 1 0
0 0 1

0 = -0.64767
1 = -0.64996
2 = -0.65619
3 = -0.65832
4 = -0.65850
5 = -0.65911
6 = -0.65918
7 = -0.65941
8 = -0.65941
-----
itk::simple::BSplineTransform
BSplineTransform (0x11ae0a2d0)
RTTI typeinfo: itk::BSplineTransform<double, 3u, 3u>
Reference Count: 3
Modified Time: 80091
Debug: Off
Object Name:
Observers:
none
CoefficientImage: [ 0x11b7ee1f0, 0x11b7f58c0, 0x11b7f4660 ]
TransformDomainOrigin: [-242.5, -242.5, -197.5]
TransformDomainPhysicalDimensions: [484.053, 484.053, 337.5]
TransformDomainDirection: 1 0 0
0 1 0
0 0 1

TransformDomainMeshSize: [10, 10, 10]
GridSize: [13, 13, 13]
GridOrigin: [-290.905, -290.905, -231.25]
GridSpacing: [48.4053, 48.4053, 33.75]
GridDirection: 1 0 0
0 1 0
0 0 1

```

```

itk::simple::BSplineTransform
BSplineTransform (0x11ae0a2d0)
  RTTI typeinfo: itk::BSplineTransform<double, 3u, 3u>
  Reference Count: 3
  Modified Time: 80091
  Debug: Off
  Object Name:
  Observers:
    none
  CoefficientImage: [ 0x11b7ee1f0, 0x11b7f58c0, 0x11b7f4660 ]
  TransformDomainOrigin: [-242.5, -242.5, -197.5]
  TransformDomainPhysicalDimensions: [484.053, 484.053, 337.5]
  TransformDomainDirection: 1 0 0
  0 1 0
  0 0 1

  TransformDomainMeshSize: [10, 10, 10]
  GridSize: [13, 13, 13]
  GridOrigin: [-290.905, -290.905, -231.25]
  GridSpacing: [48.4053, 48.4053, 33.75]
  GridDirection: 1 0 0
  0 1 0
  0 0 1

Optimizer stop condition: GradientDescentLineSearchOptimizerv4Template: Conv
ergence checker passed at iteration 9.
  Iteration: 9
  Metric value: -0.6594125093164644

```

Memorizzazione dell'output di registrazione mediante B-Spline in un'immagine NIFTI

```

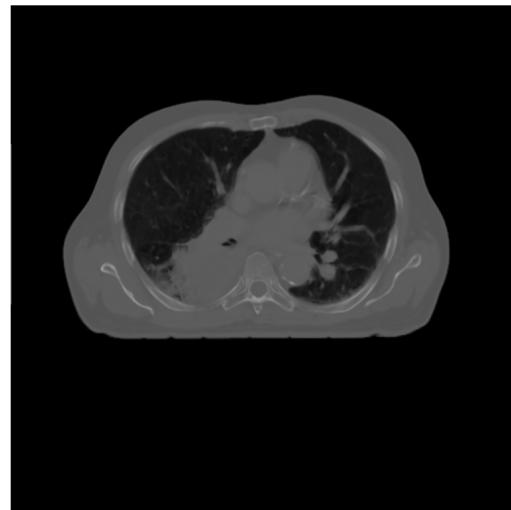
In [14]: sitk.WriteImage(cimg, "output_bspline.nii.gz")

In [37]: out_reg = sitk.ReadImage('output_bspline.nii.gz', sitk.sitkFloat32)

In [42]: #slice_out = sitk.GetArrayFromImage(out_reg)[80,:,:]
gui.MultiImageDisplay(
    image_list=[out_reg],
    title_list=["out reg"],
    figure_size=(8, 4),
    #window_level_list=[fixed_window_level, moving_window_level],
    intensity_slider_range_percentile=[0, 100],
)
VBox(children=(Box(children=(IntSlider(value=67, description='image slice:', max=135),)), Box(children=(IntRan...

```

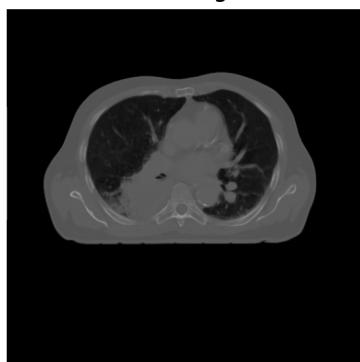
out reg



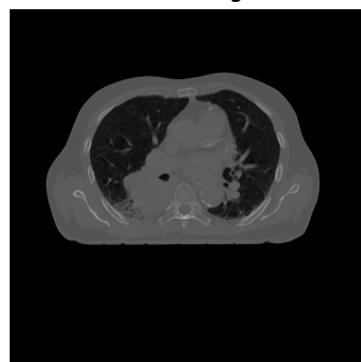
Confronto dell'output di registrazione mediante B-Spline con le immagini fixed e moving iniziali

```
In [43]: gui.MultiImageDisplay(  
    image_list=[out_reg, fixed_image, moving_image],  
    title_list=["out reg", "fixed image", "moving image"],  
    figure_size=(8, 4),  
    #window_level_list=[fixed_window_level, moving_window_level],  
    intensity_slider_range_percentile=[0, 100],  
);  
  
VBox(children=(Box(children=(IntSlider(value=67, description='image slice:',  
max=135), IntSlider(value=67, des...
```

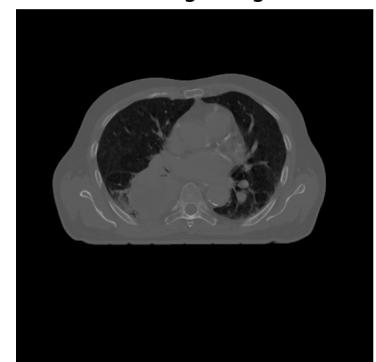
out reg



fixed image



moving image



```
In [44]: trasOut = sitk.ReadTransform('Bspline_outTxExport.tfm')  
trasOut.GetCoefficientImages()
```

```
Out[44]: (<SimpleITK.SimpleITK.Image; proxy of <Swig Object of type 'std::vector< itk::simple::Image >::value_type *' at 0x3e0bcd00>,<SimpleITK.SimpleITK.Image; proxy of <Swig Object of type 'std::vector< itk::simple::Image >::value_type *' at 0x3e0bcd350>,<SimpleITK.SimpleITK.Image; proxy of <Swig Object of type 'std::vector< itk::simple::Image >::value_type *' at 0x2f4483b40>)
```

Visualizzazione del displacement field ottenuto dalla trasformazione B-Spline mediante Slicer3D

Importiamo il file *ni.gz* salvato dalla **FFD** come volume e importiamo anche il file *.tfm* come trasformazione nel software su menzionato. Scegliamo di visualizzare il campo vettoriale secondo la modalità a griglia per le immagini in sezione, mentre il volume 3D è rappresentato mediante superfici di spostamento.

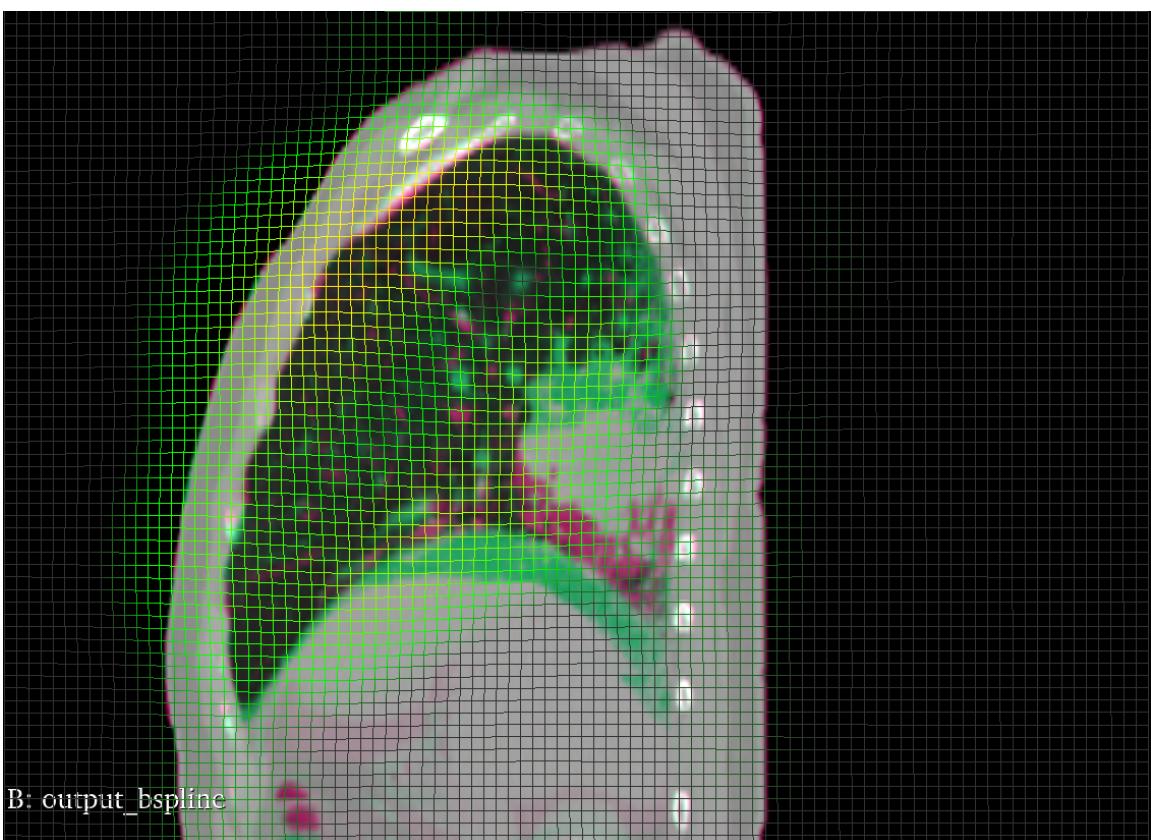
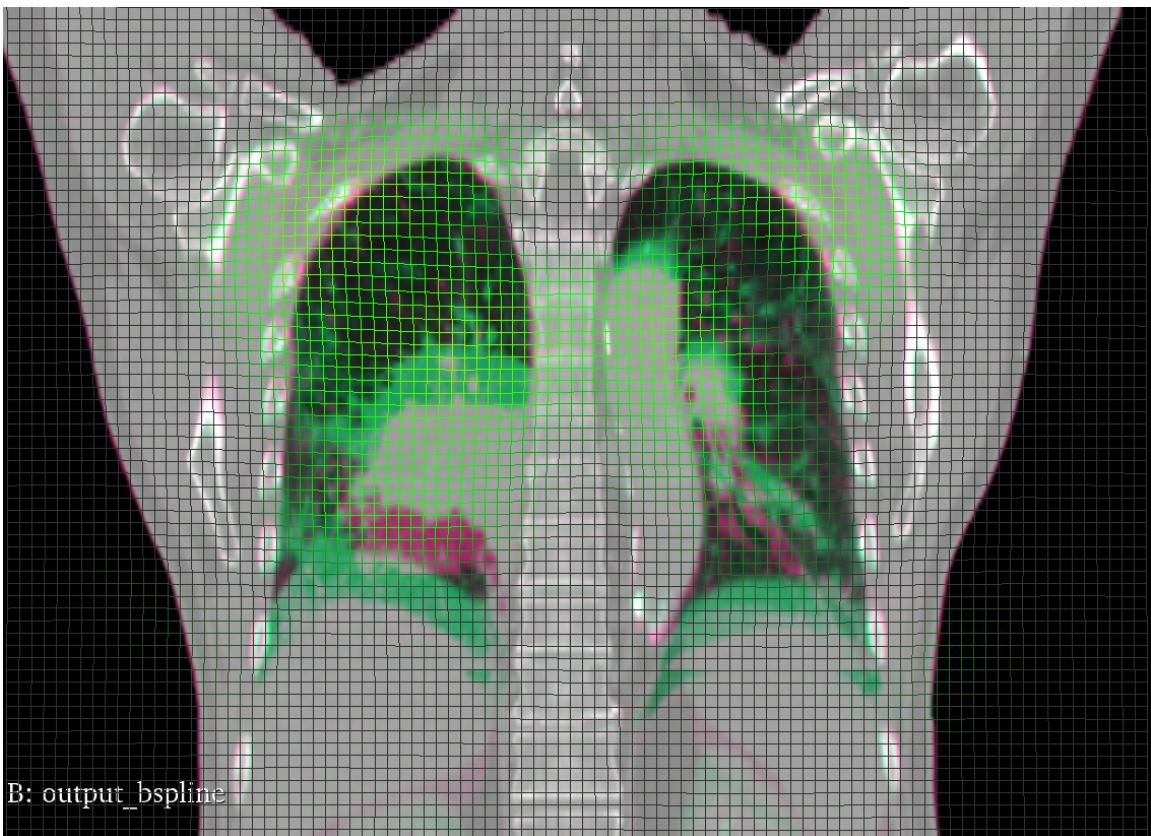
Slicer 3D è una piattaforma *open-source* per la gestione, visualizzazione, analisi scientifica ed elaborazione delle immagini medicali. Slicer permette di creare "stack di immagini", partendo da immagini DICOM, trasformandole in blocchi di dati tri, quadri e pentadimensionali (ad esempio mostrando la contemporanea variazione di due parametri, in un reticolo tridimensionale nel tempo), come in filmati 3D riguardanti la fMRI, la PET, etc. Slicer permette anche di confrontare due o più risonanze magnetiche di uno stesso soggetto nel tempo, per determinare se è apparsa una lesione (nel individuo sano), se sono apparse nuove lesioni, oppure se queste si sono ingrandite o ridotte. Viene utilizzato in molte applicazioni medicali, come per le malattie cardiovascolari, neurochirurgia, ed è anche utilizzata nella radioterapia.

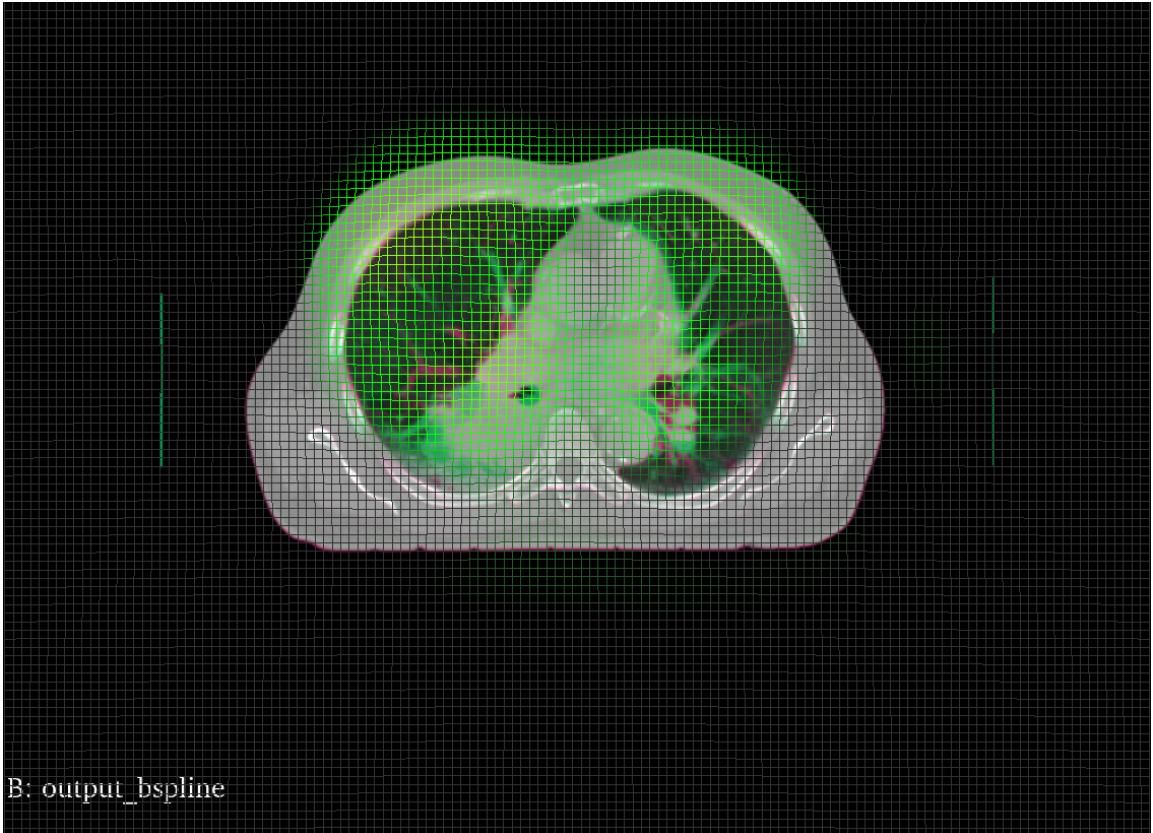
Le capacità di Slicer 3D includono:

- Capacità di leggere e di scrivere (per anonimizzare) le immagini DICOM e un buon numero di altri formati
- Visualizzazione interattiva delle immagini, triangolazione di modelli di superficie 3D, e rendering volumetrico.
- Editing manuale.
- Fusione e co-"*registering*" (fusione dei dati di due immagini in tempi diversi) usando algoritmi di trasformazione rigida e non-rigida
- Segmentazione automatica
- Analisi e visualizzazione dei dati di diffusione del tensore di imaging.
- Tracking di dispositivi per procedure mediche guidate dall'imaging.

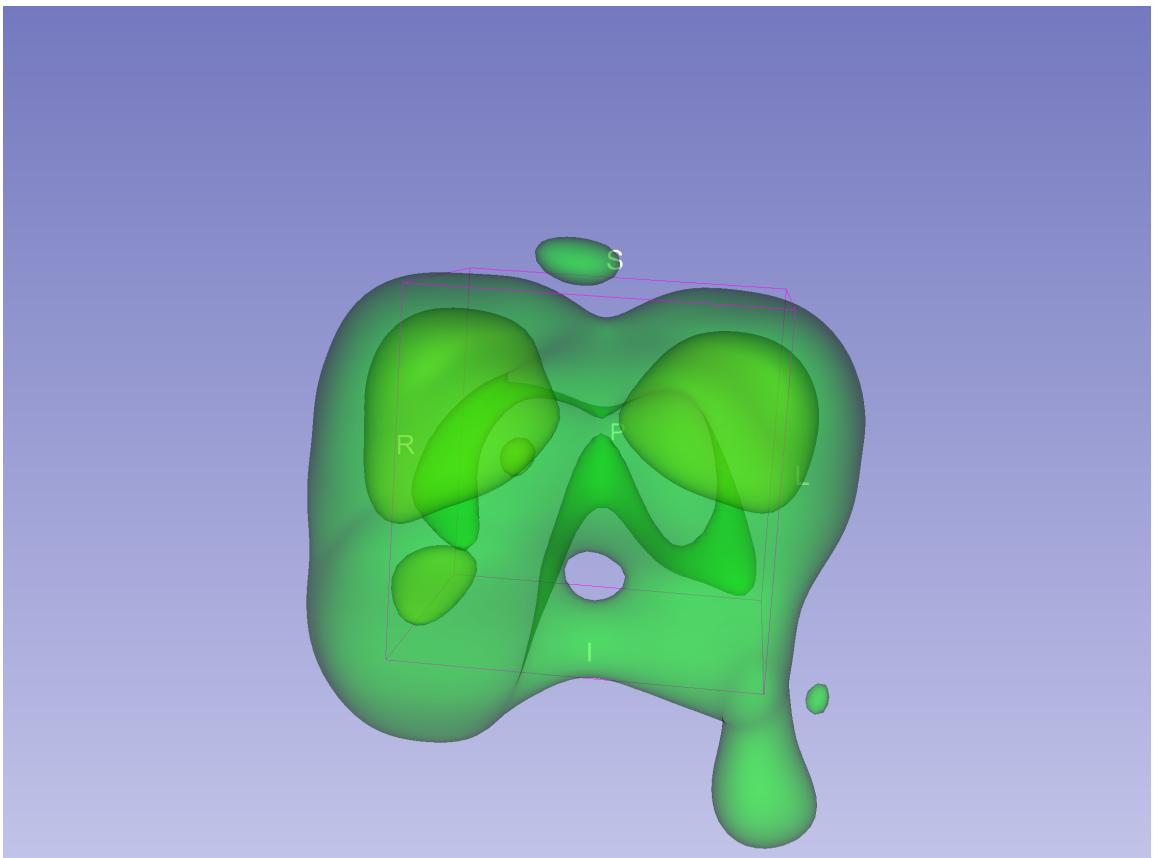
Alla fine, si avrà un vettore di spostamento medio.

- Frecce verdi : vettori di spostamento calcolati sui nodi delle Bsplines
- Frecce rosse: tutti i pixel dentro una cella che si muovono come la media dei vettori di spostamento





B: output_bspline



In []:

FFD Demons Registration (SimpleITK)

Cenni teorici al metodo Demons

Il metodo **Demons** consiste in una trasformazione non rigida di due o più immagini che sfrutta il concetto di diffusione. Il concetto teorico che sta alla base del metodo è quello dei "demoni" ovvero flussi effettori che possono diffondere attraverso i bordi fisici degli oggetti rappresentati nelle immagini. La diffusione viene guidata dalla variazione di intensità dei bordi. Il metodo Demon prevede l'uso di una immagine che si intende trasformare (moving) che sarà quindi in movimento e di una fissa da usare come riferimento (fixed). Questo porta ad una elevata alterazione dell'immagine moving o, nel caso di un approccio multimmagine, dell'intero set di TC del soggetto in esame. Per questo viene introdotto nel metodo un algoritmo bilanciato, che, iterativamente, applica le deformazioni del Demons ai due set di immagini in modo alternato. La deformazione finale si ottiene ad ogni iterazione combinando la deformazione diretta sia della fixed che della moving. L'applicazione del metodo Demon permette di ottenere, partendo da due set di immagini, un unico campo vettoriale degli spostamenti, indicato come griglia di deformazione, che permette la trasformazione della TC del soggetto.

Output ottenuto alla fine della registrazione

Alla fine della registrazione e dell'esecuzione di tutti gli script presenti in questo paragrafo del notebook, avremo due file di output:

trasformazione.tfm
cimg.nii.gz

La prima rappresenta la trasformazione della registrazione B-Spline, mentre la seconda consiste nell'intero volume registrato mediante free form deformation.

```
In [3]: fixed = sitk.ReadImage("./chest4D/chest0.nrrd", sitk.sitkFloat32)
moving = sitk.ReadImage("./chest4D/chest5.nrrd", sitk.sitkFloat32)

In [8]: def command_iteration(filter):
    print(f"{filter.GetElapsedIterations():3} = {filter.GetMetric():10.5f}")

    matcher = sitk.HistogramMatchingImageFilter()
    matcher.SetNumberOfHistogramLevels(1024)
    matcher.SetNumberOfMatchPoints(7)
    matcher.ThresholdAtMeanIntensityOn()
    moving = matcher.Execute(moving, fixed)

    # The basic Demons Registration Filter
    # Note there is a whole family of Demons Registration algorithms included in
    # SimpleITK
    demons = sitk.DemonsRegistrationFilter()
    demons.SetNumberOfIterations(50)
    # Standard deviation for Gaussian smoothing of displacement field
    demons.SetStandardDeviations(1.0)

    demons.AddCommand(sitk.sitkIterationEvent, lambda: command_iteration(demons))

    displacementField = demons.Execute(fixed, moving)

    print("-----")
    print(f"Number Of Iterations: {demons.GetElapsedIterations()}")
    print(f" RMS: {demons.GetRMSChange()}"
```

```
outTx = sitk.DisplacementFieldTransform(displacementField)

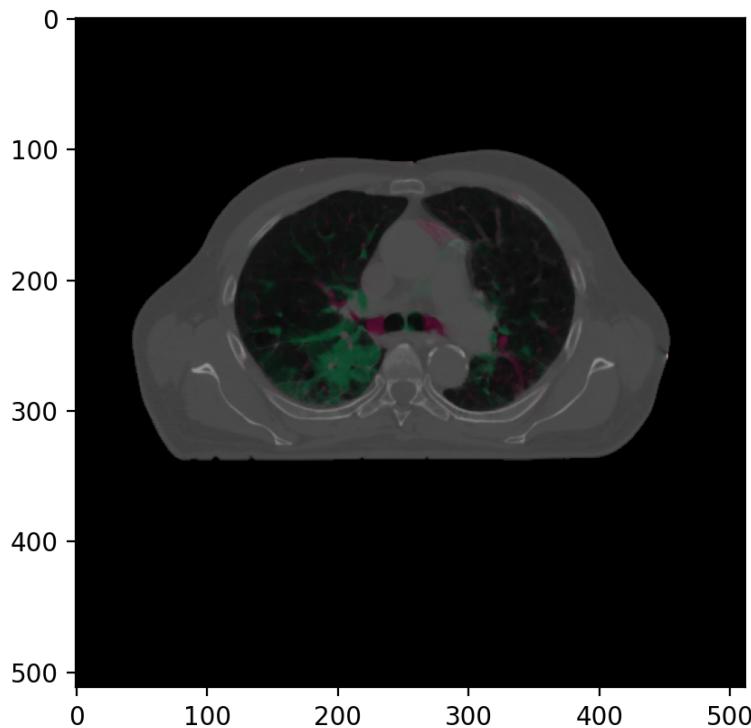
sitk.WriteTransform(outTx, 'trasformazione.tfm')

if "SITK_NOSHOW" not in os.environ:
    resampler = sitk.ResampleImageFilter()
    resampler.SetReferenceImage(fixed)
    resampler.SetInterpolator(sitk.sitkLinear)
    resampler.SetDefaultPixelValue(100)
    resampler.SetTransform(outTx)

    out = resampler.Execute(moving)
    simg1 = sitk.Cast(sitk.RescaleIntensity(fixed), sitk.sitkUInt8)
    simg2 = sitk.Cast(sitk.RescaleIntensity(out), sitk.sitkUInt8)
    # Use the // floor division operator so that the pixel type is
    # the same for all three images which is the expectation for
    # the compose filter.
    cimg = sitk.Compose(simg1, simg2, simg1 // 2.0 + simg2 // 2.0)
    sitk.Show(cimg, "DeformableRegistration1 Composition")
```

```
1 = 16168.53646
2 = 14398.95175
3 = 13320.93716
4 = 12549.17583
5 = 11904.49175
6 = 11362.41621
7 = 10902.54154
8 = 10504.21507
9 = 10154.24120
10 = 9843.35863
11 = 9564.98668
12 = 9313.52283
13 = 9085.83243
14 = 8878.81876
15 = 8690.73041
16 = 8518.91070
17 = 8362.68629
18 = 8220.20084
19 = 8089.68869
20 = 7970.16719
21 = 7860.29704
22 = 7759.65557
23 = 7666.22517
24 = 7579.72605
25 = 7498.61473
26 = 7422.94564
27 = 7351.57280
28 = 7284.61319
29 = 7221.06074
30 = 7161.11601
31 = 7103.95143
32 = 7049.87413
33 = 6998.17501
34 = 6949.17500
35 = 6901.98435
36 = 6856.97668
37 = 6813.52112
38 = 6771.94253
39 = 6731.68262
40 = 6693.03579
41 = 6655.58538
42 = 6619.79891
43 = 6584.80984
44 = 6551.30498
45 = 6518.49771
46 = 6486.87349
47 = 6455.83024
48 = 6425.76942
49 = 6396.11740
50 = 6367.40719
-----
Number Of Iterations: 50
RMS: 0.28453761070269307
```

```
In [5]: # sitk.GetArrayFromImage(cimg)[z,y,x]
slice = sitk.GetArrayFromImage(cimg)[80,:,:]
imshow(slice)
```



Out[5]: <matplotlib.image.AxesImage at 0x1600a6020>

Visualizzazione dell'output di registrazione mediante DemonRegistrationFilter

```
In [9]: sitk.WriteImage(cimg, 'cimg.nii.gz')
image = sitk.ReadImage('cimg.nii.gz')
```

```
In [12]: fetta = sitk.GetArrayViewFromImage(image)[80,:,:]
imshow(fetta)
```

Out[12]: <matplotlib.image.AxesImage at 0x28e1f9900>

Calcolo e visualizzazione del determinante Jacobiano della trasformazione

Utilizzando la funzione `DisplacementFieldJacobianDeterminant` incluso in `SimpleITK` si calcola un'immagine scalare a partire da un'immagine vettoriale (ad esempio, un campo di deformazione) in ingresso, dove ogni scalare di uscita, in ogni pixel, è il determinante jacobiano del campo vettoriale in quella posizione.

Questo calcolo è corretto nel caso in cui l'immagine vettoriale sia uno "spostamento" dalla posizione corrente. Il calcolo del determinante jacobiano è dato da:

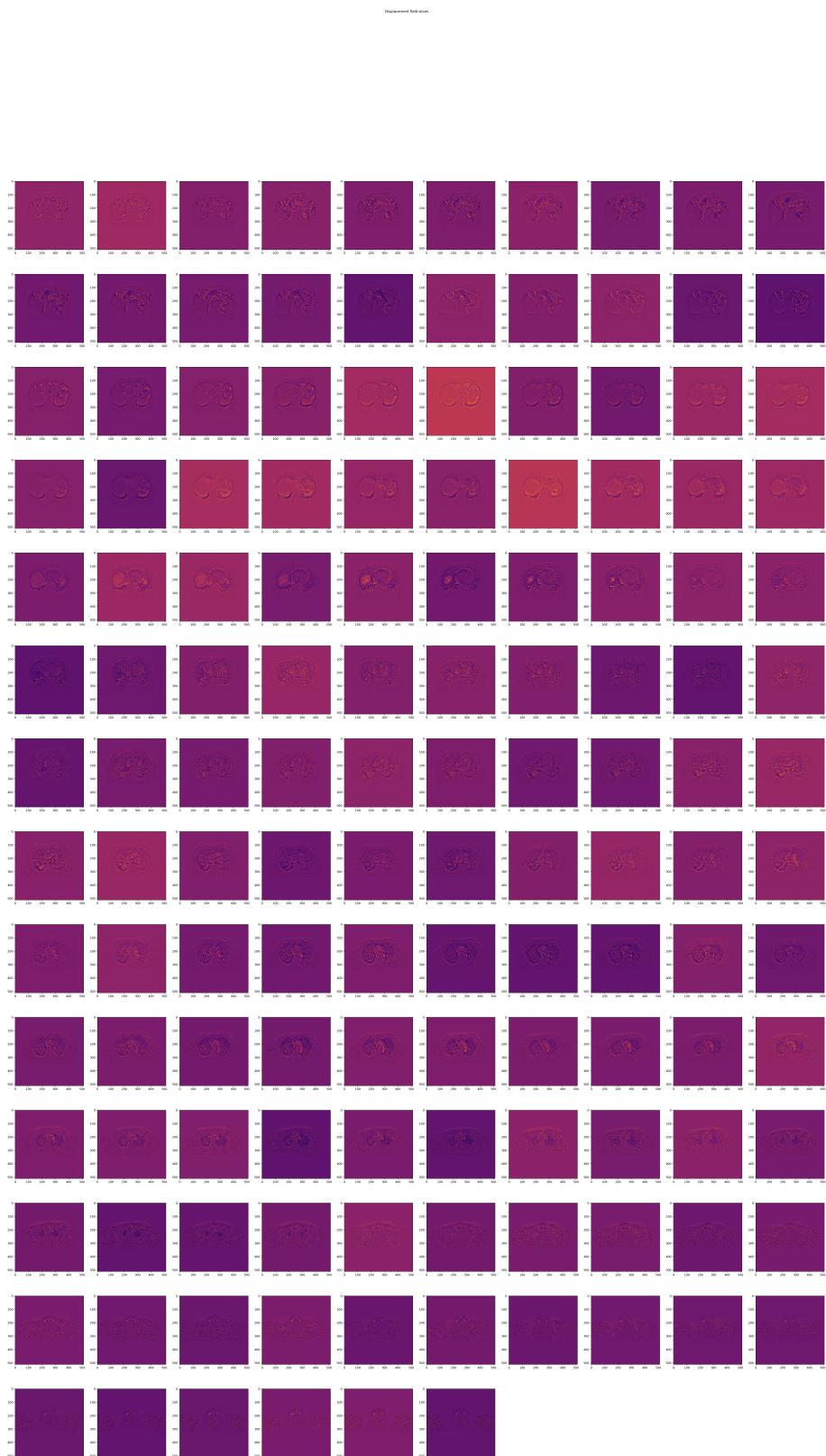
$$DET \frac{dT}{dx} = DET \frac{I + du}{dx}$$

Si noti che il determinante di un campo vettoriale nullo è anch'esso nullo, mentre il determinante jacobiano della corrispondente trasformazione di deformazione identica è

1,0. Per calcolare la deformazione effettiva il determinante jacobiano 1,0 deve essere aggiunto agli elementi diagonali della jacobiana prima di prendere la derivata

Questo filtro ha un parametro richiesto che definisce il tipo di immagine in ingresso. Si presume che il tipo di pixel dell'immagine in ingresso sia un vettore (ad esempio, `itk::Vector`, `itk::RGBPixel`, `itk::FixedArray`). Il tipo scalare dei componenti del vettore deve essere `Castable` in virgola mobile.

```
In [13]: jb = sitk.DisplacementFieldJacobianDeterminant(outTx.GetDisplacementField())  
  
In [14]: #To show all the slices using the same color range you need to get the min  
#and max of the whole volume and use matplotlib and numpy for display  
# use matplotlib and numpy doc in order to get min and max  
#of all images and normalize them in one range  
  
jacobian_det_np_arr = sitk.GetArrayViewFromImage(jb)  
  
#inizializzo un array vuoto  
images = []  
# inizializzo un'immagine 50x50  
fig = plt.figure(figsize=(50,80)) # width, height in inches  
fig.suptitle('Displacement field slices')  
# Itero in tutte le slice  
for i in range(1,jacobian_det_np_arr.shape[0]+1):  
    dat = jacobian_det_np_arr[i-1,:,:,:]  
    images.append(dat)  
    sub = fig.add_subplot(14, 10, i)  
    sub.imshow(jacobian_det_np_arr[i-1,:,:,:], interpolation='nearest',
```



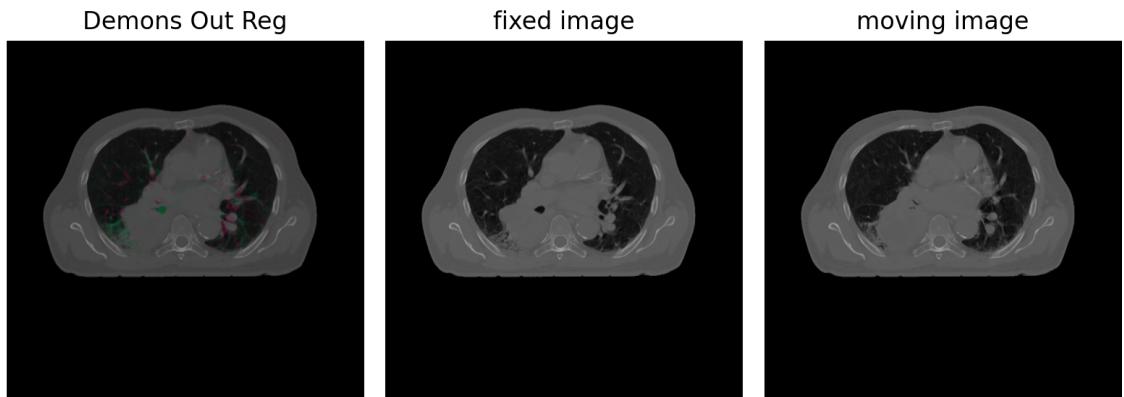
```
In [15]: gui.MultiImageDisplay(  
    image_list=[image, fixed, moving],  
    title_list=["Demons Out Reg", "fixed image", "moving image"],  
    figure_size=(8, 4),  
    #window_level_list=[fixed_window_level, moving_window_level],
```

```

    intensity_slider_range_percentile=[0, 100],
);

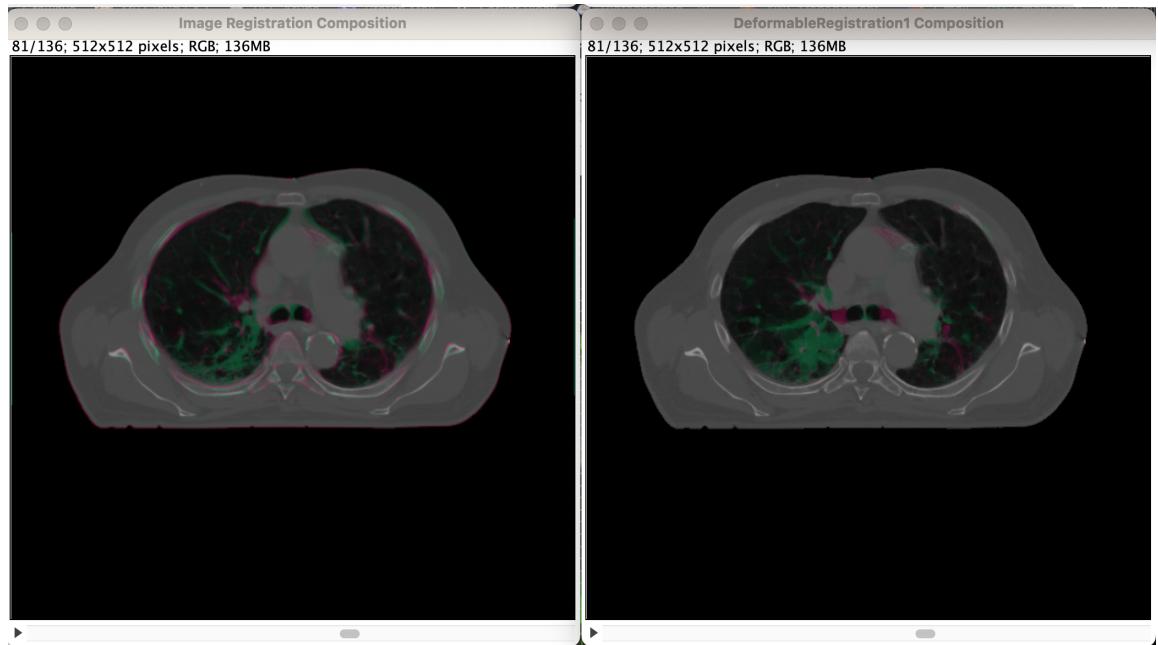
VBox(children=(Box(children=(IntSlider(value=67, description='image slice:', max=135), IntSlider(value=67, des...

```



Le due registrazioni ottenute con SimpleITK a confronto

1. Output proveniente dalla trasformazione Free Form **B-Spline**
2. Output della trasformazione utilizzando il **DemonsRegistrationFilter**



Confronto con la registrazione effettuata su macchina virtuale e Plastimatch configurato

È stata effettuata la medesima registrazione mediante B-Spline utilizzando anche il software **Plastimatch** configurata all'interno della macchina virtuale. Il risultato in output è stato quindi inserito all'interno del software di Slicer3D per la visualizzazione del vettore di spostamento generato dalla trasfromazione.

In []:

In []:

In []:

Visualizzazione del Dispacement Field generato dalla registrazione in Plastimatch

Il campo vettoriale effettua la media delle bspline su ogni voxel presente nell'immagine 4D ciò ci permette di osservare anatomicamente come si muove il sito anatomico che stiamo andando ad analizzare. In questo caso è stato preso in esame il torace di un paziente. Le immagini utilizzate sono in formato nrrd e sono state utilizzate chest_0 come fixed e chest_5 come moving. Grazie all'utilizzo di un file txt denominato reg_par, che è appunto l'abbreviazione di registration parameter, e che contiene le righe scritte qui di seguito:

```
[GLOBAL]
fixed=chest0.nrrd
moving=chest5.nrrd
vf_out=out_vf.nrrd
xform_out=out_xform.txt
img_out=out_img.nrrd

[COMMENT]
xform=rigid
optim=versor
impl=plastimatch
metric=mse
max_its=100
convergence_tol=3
grad_tol=1
res=2 2 2

[STAGE]
xform=bspline
optim=lbfgsb
impl=plastimatch
res=1 1 1
grid_spac=35 35 35
max_its=100
#regularization_lambda=0.0005

[STAGE]
xform=bspline
optim=lbfgsb
impl=plastimatch
res=1 1 1
grid_spac=10 10 20
max_its=100
#regularization_lambda=0.0005
```

Sono stati inseriti nei vari campi il nome dell'immagine fixed, il nome dell'immagine moving e in uscita è stato ottenuto un file nrrd contenente il vector field più un file txt contenente tutti i dati inerenti alla trasformazione outxform. Grazie a plastimatch quindi è stato possibile effettuare questa registrazione ed ottenere il vector field che contiene dei vettori 4D. Per visualizzare al meglio il campo vettoriale è stato impiegato slicer grazie al quale, dopo aver caricato le immagini fixed e moved e successivamente anche il file nrrd contenente il campo vettoriale, è stato possibile analizzare l'andamento del sito anatomico osservato. Si noti che nell'immagine seguente i vettori tendono ad essere rivolti verso il basso, con diverse intensità. Questo è ragionevole poichè analizzando un torace, abbiamo che, nella fase di inspirazione, il diaframma si abbassa per dare modo ai polmoni di espandersi.

Oltre al precedente vector field che è stato ottenuto utilizzando una rappresentazione grafica con il software 3D slicer, ne è stato ottenuto un altro utilizzando sempre le registrazioni ottenute tramite `plastimatch`, solo che al posto di utilizzare slicer è stata utilizzata una funzione di `matplotlib` chiamata `quiver`. Per fare ciò è stato creato un ambiente virtuale di python che una volta attivato ci ha permesso di implementare alcune linee di codice con le quali abbiamo importato tutte le librerie a noi utili per il calcolo del vector field.

Sfrutto la libreria `matplotlib.pyplot.quiver` per la visualizzazione di campi vettoriali in 2D

```
In [51]: # Importo nuovamente i file dalla cartella in locale ./chest4D per non dover
fixed_image = sitk.ReadImage("./chest4D/chest0.nrrd")
fixed_np = sitk.GetArrayFromImage(fixed_image)

moving_image = sitk.ReadImage("./chest4D/chest5.nrrd", sitk.sitkFloat32)
moving_np = sitk.GetArrayFromImage(moving_image)

# Importo anche i file in output forniti dalla registrazione in Plastimatch
out_vf = sitk.ReadImage("./guido/out_vf.nrrd")
outvf_np = sitk.GetArrayFromImage(out_vf)
out_img = sitk.ReadImage("./guido/out_img.nrrd")
outimg_np = sitk.GetArrayFromImage(out_img)
```

```
In [52]: outvf_np.shape
```

```
Out[52]: (136, 512, 512, 3)
```

```
In [71]: #Visualizzazione nel piano trasverso del displacement field
plt.figure(figsize=(80, 80))

det = sitk.GetArrayFromImage(sitk.DisplacementFieldJacobianDeterminant(out_vf))
vectors = sitk.GetArrayFromImage(out_vf)[85,:,:]

# det.shape = 512
x = y = 512

#X, Y = np.meshgrid(512,512)
#vectors = sitk.GetArrayFromImage(out_vf)[85,:,:]

X, Y = np.meshgrid(np.arange(0, x, 1), np.arange(0, y, 1), indexing='ij')

U = vectors[:, :, 0]
```

```

V = vectors[:, :, 1]

plt.imshow(det, cmap='jet')

plt.quiver(Y, X, V, U, angles='xy', scale_units='xy', color='white', headlen=5,
           pivot='mid', scale=1)

```

Out[71]: <matplotlib.quiver.Quiver at 0x307e6f190>



Riduco la densità delle arrows presenti nell'immagine precedente

```

In [74]: # integro una variabile step per definire il numero di arrow per ogni coordinate
plt.figure(figsize=(80, 80))

det = sitk.GetArrayFromImage(sitk.DisplacementFieldJacobianDeterminant(out_vf))
vectors = sitk.GetArrayFromImage(out_vf)[85, :, :]

# det.shape = 512
x = y = 512

#X, Y = np.meshgrid(512, 512)
#vectors = sitk.GetArrayFromImage(out_vf)[85, :, :]

X, Y = np.meshgrid(np.arange(0, x, 1), np.arange(0, y, 1), indexing='ij')

U = vectors[:, :, 0]

```

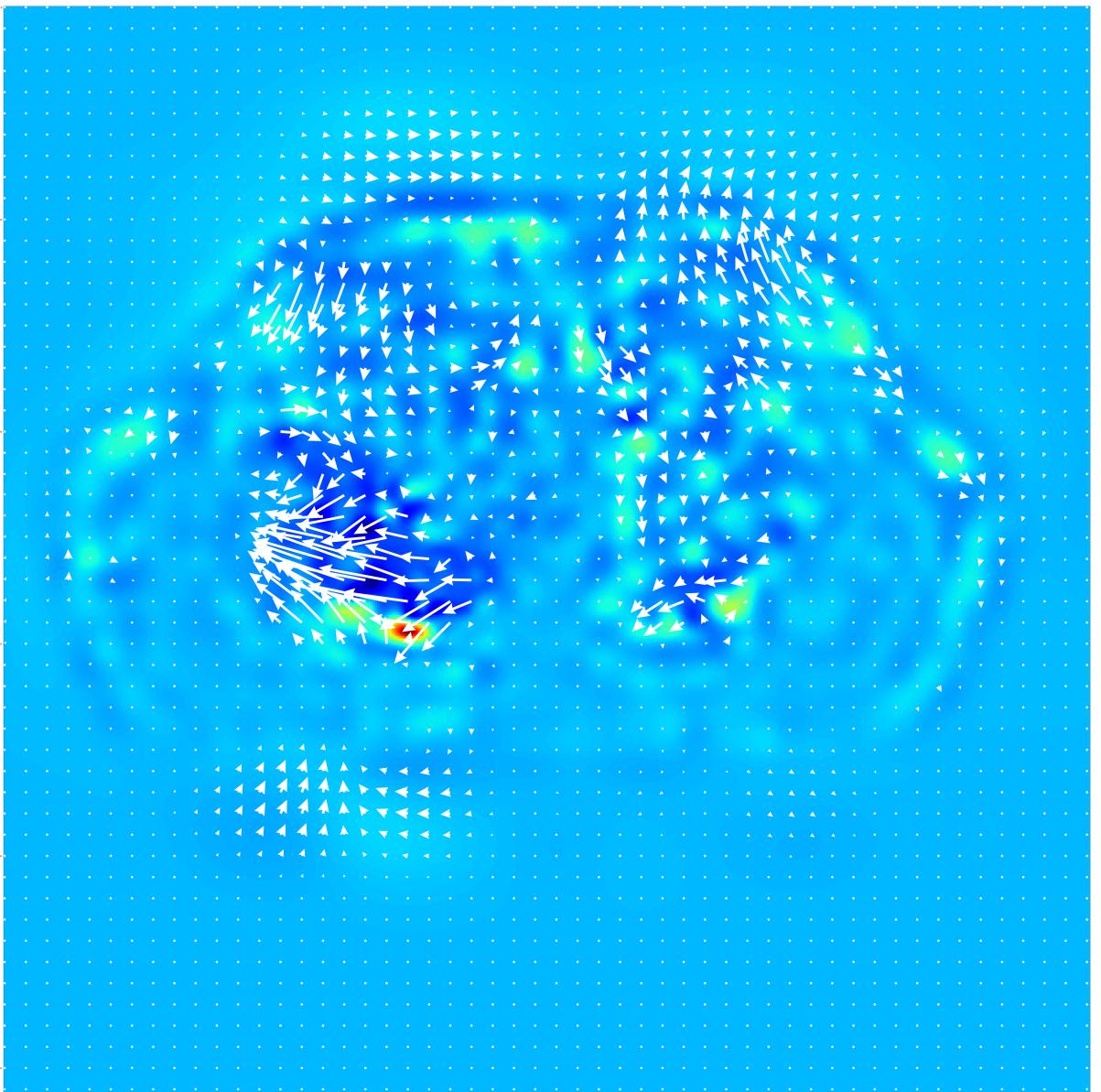
```
V = vectors[:, :, 1]

step= 10

plt.imshow(det, cmap='jet')

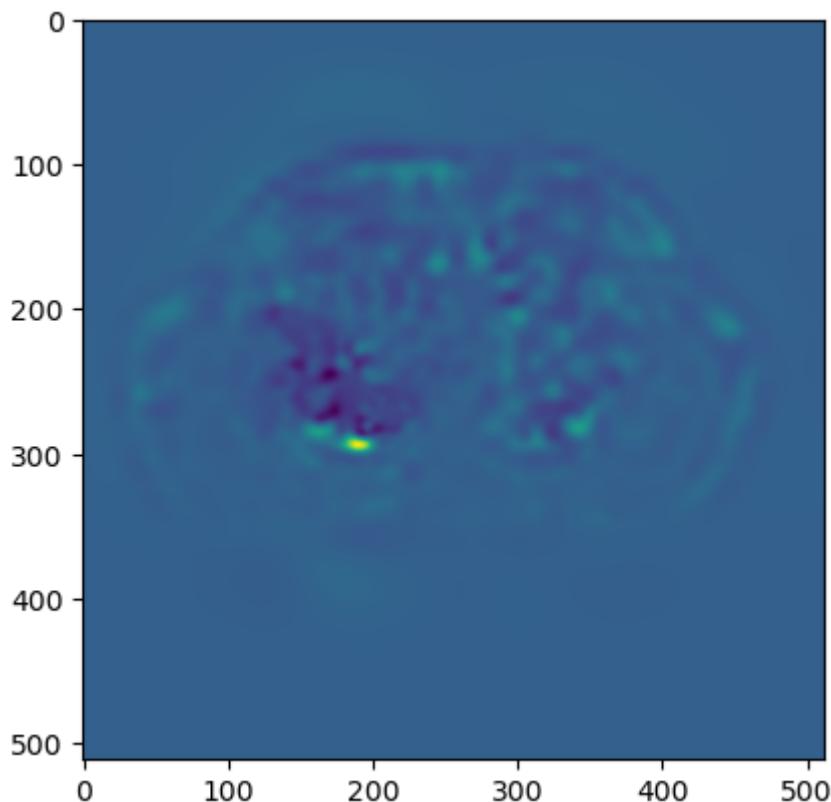
plt.quiver(Y[::step,::step], X[::step,::step], V[::step,::step], U[::step,::step],
           angles='xy', scale_units='xy', color='white', headlength=5, headwidth=2,
           scale=1)
```

Out[74]: <matplotlib.quiver.Quiver at 0x3998331f0>



In [57]: *#mostro a video il determinante jacobiano del displacement field generato da imshow(det)*

Out[57]: <matplotlib.image.AxesImage at 0x3583ce140>



Vista coronale (frontale) del displacement field

```
In [75]: plt.figure(figsize=(80, 80))

pixel_ratio=out_img.GetSpacing()[2]/out_img.GetSpacing()[0]

det = sitk.GetArrayFromImage(sitk.DisplacementFieldJacobianDeterminant(out_v
vectors = sitk.GetArrayFromImage(out_vf)[::-1,250,:]

# det.shape = 512
x, y = det.shape

#X, Y = np.meshgrid(512,512)
#vectors = sitk.GetArrayFromImage(out_vf)[85,:,:]

X, Y = np.meshgrid(np.arange(0, x, 1), np.arange(0, y, 1), indexing='ij')

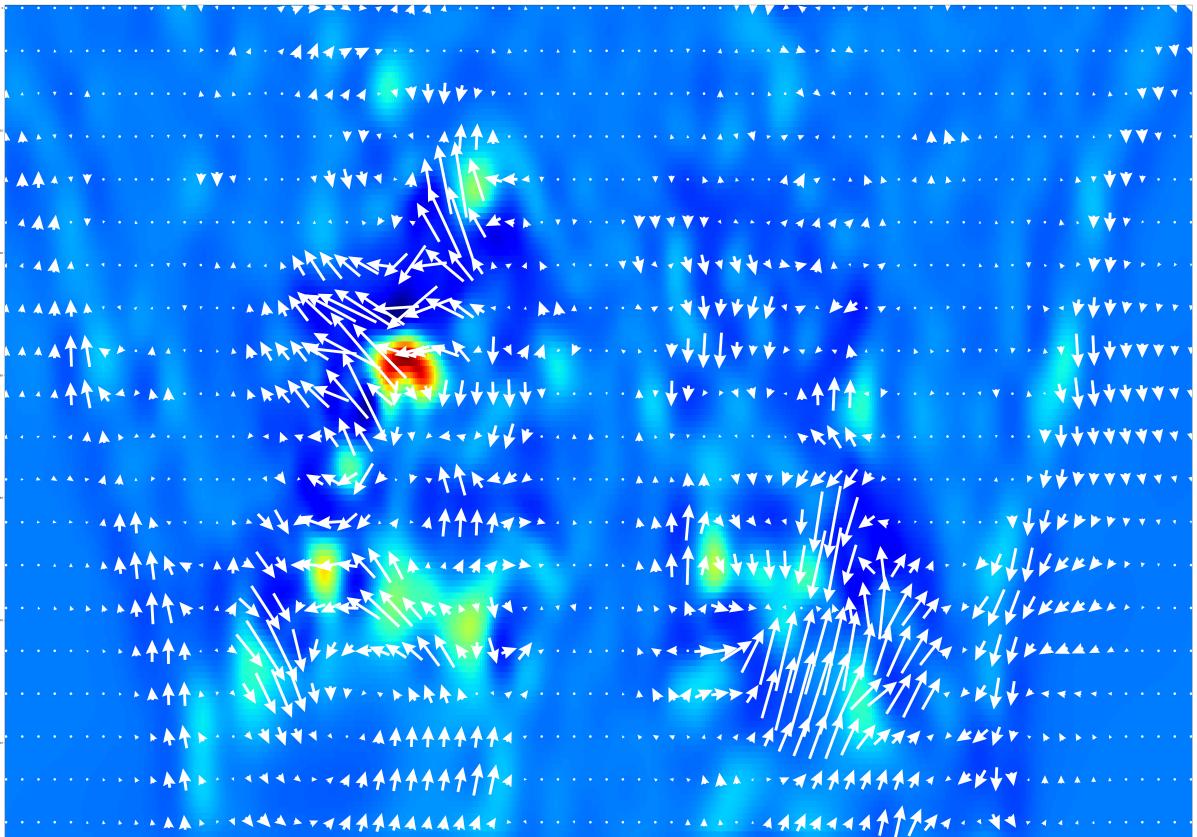
U = vectors[:, :, 0]
V = vectors[:, :, 1]

step=7

plt.imshow(det, cmap='jet', aspect=pixel_ratio)

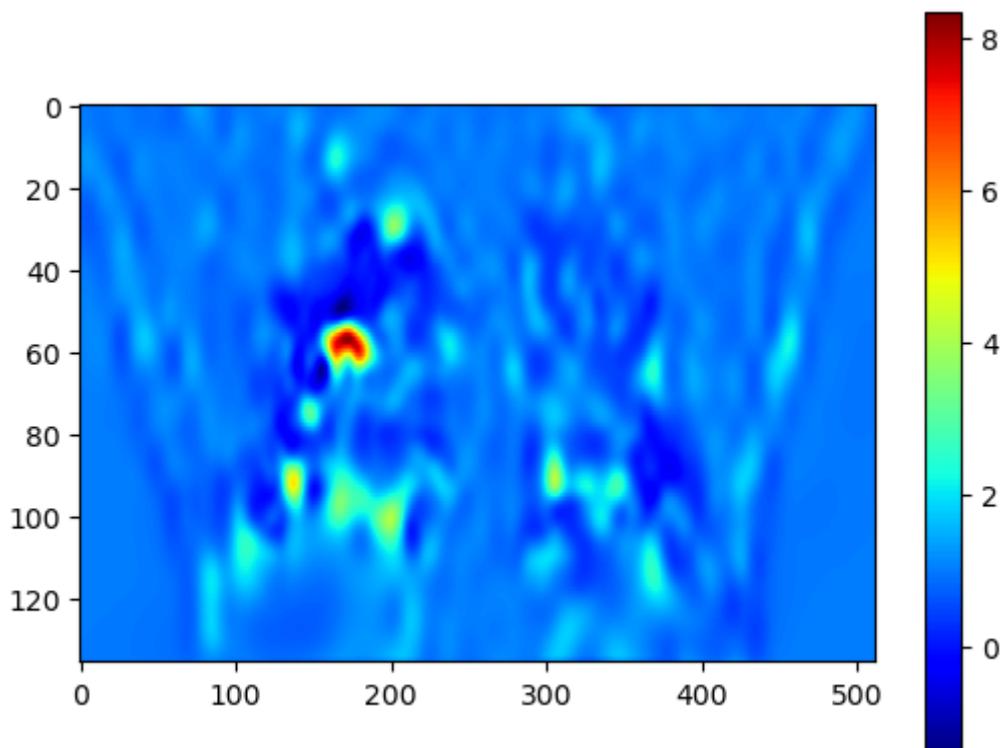
plt.quiver(Y[::-step, ::step], X[::-step, ::step], V[::-step, ::step],
            angles='xy', scale_units='xy', color='white', scale=1, headlength=0,
            pivot='mid')
```

```
Out[75]: <matplotlib.quiver.Quiver at 0x3708c7e20>
```



```
In [59]: imshow(det, aspect=pixel_ratio, cmap='jet')
plt.colorbar()
```

```
Out[59]: <matplotlib.colorbar.Colorbar at 0x35a67a470>
```



**Mostro per ogni slice del volume un'immagine di valori scalari
(`sitk.DisplacementFieldJacobianDeterminant`)**

```
In [61]: np_displacement_field = sitk.GetArrayFromImage(out_vf)
#Convert the numpy array to a 256x256x20 image with each pixel
#being a 3D vector and compute the jacobian determinant volume
```

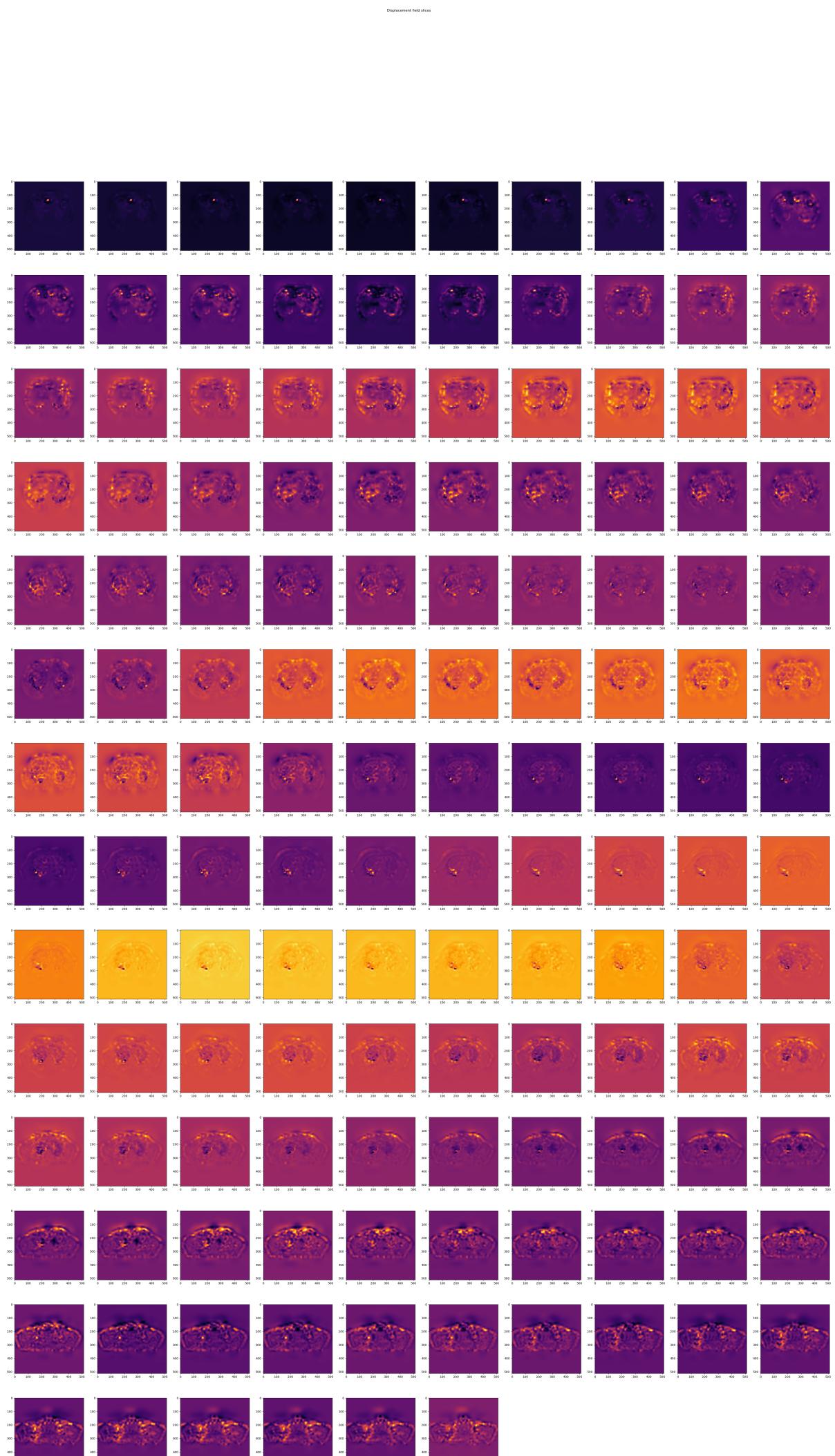
```
sitk_displacement_field = sitk.GetImageFromArray(np_displacement_field, isVector=1)
jacobian_det_volume = sitk.DisplacementFieldJacobianDeterminant(sitk_displacement_field)

#To show all the slices using the same color range you need to get the min
#and max of the whole volume and use matplotlib and numpy for display
jacobian_det_np_arr = sitk.GetArrayViewFromImage(jacobian_det_volume)

#inizializzo un array vuoto
images = []
# inizializzo un'immagine 50x50
fig = plt.figure(figsize=(50,80)) # width, height in inches
fig.suptitle('Displacement field slices')

# Iterating over the slices
for i in range(1,jacobian_det_np_arr.shape[0]+1):
    dat = jacobian_det_np_arr[i-1,:,:,:]
    images.append(dat)
    sub = fig.add_subplot(14, 10, i)
    sub.imshow(jacobian_det_np_arr[i-1,:,:,:], interpolation='nearest', cmap="jet")

plt.savefig("displacement_field_over_slices_trasversal_view.png")
```



```
[3]:
```

```
In [62]: np_displacement_field = sitk.GetArrayFromImage(out_vf)
#Convert the numpy array to a 256x256x20 image with each pixel
#being a 3D vector and compute the jacobian determinant volume
sitk_displacement_field = sitk.GetImageFromArray(np_displacement_field, isVector=True)
jacobian_det_volume = sitk.DisplacementFieldJacobianDeterminant(sitk_displacement_field)

#To show all the slices using the same color range you need to get the min
#and max of the whole volume and use matplotlib and numpy for display
jacobian_det_np_arr = sitk.GetArrayViewFromImage(jacobian_det_volume)

#inizializzo un array vuoto
images = []
# inizializzo un'immagine 50x50
fig = plt.figure(figsize=(50,80)) # width, height in inches
fig.suptitle('Displacement field slices')

# Iterating over the slices
for i in range(1,jacobian_det_np_arr.shape[0]+1):
    dat = jacobian_det_np_arr[::-1,i-1,:]
    images.append(dat)
    sub = fig.add_subplot(14, 10, i)
    sub.imshow(jacobian_det_np_arr[::-1,i-1,:], interpolation='nearest', cmap='jet')

plt.savefig("displacement_field_over_slices_coronal_view.png")
```

