Augmented Lagrangian Digital Image Correlation (AL-DIC) Code Manual (v3.3)

Jin Yang[1,2†], Kaushik Bhattacharya[2‡]

[1] Department of Mechanical Engineering, University of Wisconsin-Madison
[2] Division of Engineering and Applied Science, California Institute of Technology

Email: [†]jyang526@wisc.edu; [‡]bhatta@caltech.edu

Github page: https://github.com/jyang526843/2D_ALDIC
MATLAB FileExchange page: https://www.mathworks.com/matlabcentral/fileexchange/70499-augmented-lagrangian-digital-image-correlation-and-tracking

Last time updated on October 21, 2020

Contents

1 Introduction

Digital image correlation (DIC) technique is a powerful experimental tool for measuring full-field displacements and strains. Most current DIC algorithms can be categorized into either *local* or finite-element-based *global* methods, see Fig. 1. As with most experimental approaches, there are drawbacks with each of these methods. In the local method the subset deformations are estimated independently and the computed displacement field may not necessarily be kinematically compatible. Thus, the deformation gradients can be noisy, especially when using small subsets. Although the global method often enforces kinematic compatibility, it generally incurs substantially greater computational costs than its local counterpart, which is especially significant for large data sets. Here we present a new hybrid DIC algorithm, called *augmented Lagrangian digital image correlation (ALDIC)* [1], which combines the advantages of both the local (fast computation times) and global (compatible displacement field) methods.

ALDIC code is freely available at Github and MATLAB File Exchange (link: [2]) and solves the general motion optimization problem by using the alternating direction method of multipliers (ADMM) [3].

We demonstrate that our ALDIC algorithm has high accuracy and precision while maintaining low computational cost, and is a significant improvement compared to current local and global DIC methods.

For a review of both local and global DIC methods, and details of this new proposed ALDIC method, please see our paper [1] (full text can also be requested at [4]).

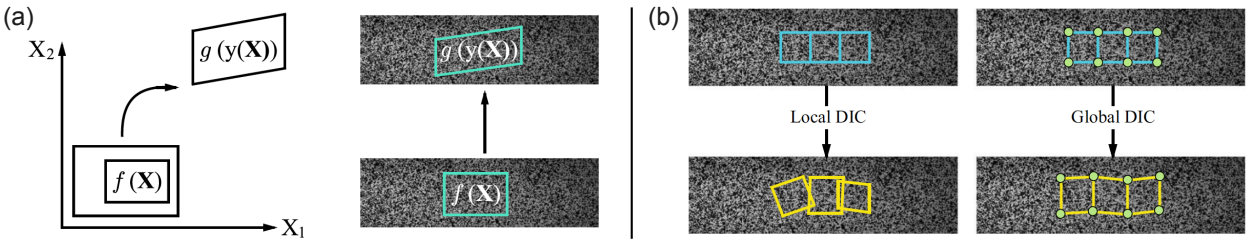


Figure 1: (a) Schematic showing a DIC reference image f(**X**), with a general speckle pattern, deforming into the deformed image g(**y**(**X**)) under some mapping **y**. **X** and **y** coordinates are in the reference and deformed images, respectively. (b) A schematic comparison between the local DIC method (left), where all the subsets are analyzed independently, and the finite-element-based global DIC method (right), where a global basis set is used to represent the full-field deformation.

Here some advantages of ALDIC algorithm are highlighted:

(i) It's a fast algorithm using distributed parallel computing for a global nonconvex optimization.

(ii) Global kinematic compatibility is added as a global constraint in the form of augmented Lagrangian, and solved using Alternating Direction Method of Multipliers (ADMM) scheme.

(iii) Both displacement fields and affine deformation gradients are correlated at the same time.

(iv) No need of much art-of-work about choosing displacement smoothing filters.

(v) It works well with compressed images and adaptive mesh [5, 6].

(vi) It can solve an image sequence with multiple frames. Both *cumulative* and *incremental* DIC modes are implemented where the latter one is quite useful for measuring very large deformations.

(vii) If you are applying digital volume correlation, please refer to our ALDVC paper [7, 8] and code [9, 10].

2 Code installation

ALDIC code can be downloaded at [2]. It has been tested on MATLAB versions later than R2018a. Parallel Computing Toolbox is highly recommended but not necessary to speed up the code.

To install the code, please download and unzip the code folder and put this folder on the MATLAB current working path.

After opening the main file "main_ALDIC.m", as shown in Fig. 2, ALDIC code can be executed by each section. Once you are familiar with ALDIC code, you can execute the whole main file by clicking the "Run" button to run the whole file at one time ("EDITOR >> RUN >>  "). ALDIC code is easy to modify based on user's custom parameter choice. In this code manual, we will introduce ALDIC code section by section.

3 Code Section 1. MATLAB mex setup

Execute this section and we will try to build "mex" functions from C/C++ source codes for image grayscale value interpolation, where linear, bi-cubic (by default) and bi-cubic splines interpolations are implemented in this code. For example, by default we use bi-cubic interpolations where the associated mex set up file is called "ba_interp2.cpp" [11].

3.1 Test MATLAB mex setup

First, we test whether there is already a C/C++ compiler installed on your computer by inputting `mex -setup` and press Enter key on the MATLAB command window. If an available C/C++ compiler is already installed, please skip Section 3.2 and jump to Section 3.3.

3.2 Install mex C/C++ compiler

The step of installing mex C/C++ compilers is a common step for users to run C/C++ codes with MATLAB. Mac users usually don't come across the error message from mex C/C++ compilers. For Windows users, you can follow these steps to install mex C/C++ compiler. More details can be found in [12, 13].

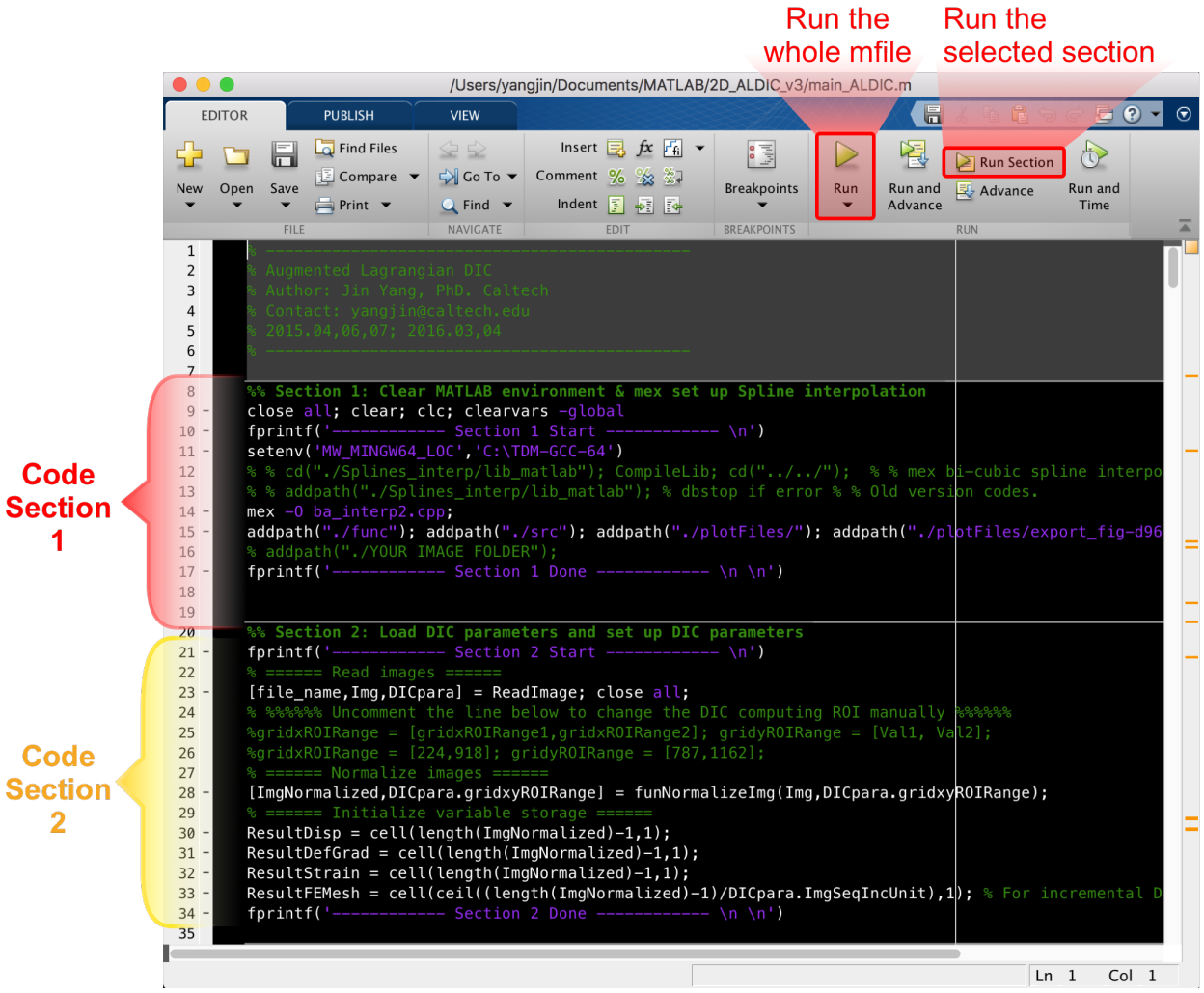


Figure 2: Main file of ALDIC code: "main_ALDIC.m". Each section can be executed in order by clicking "Run Section".

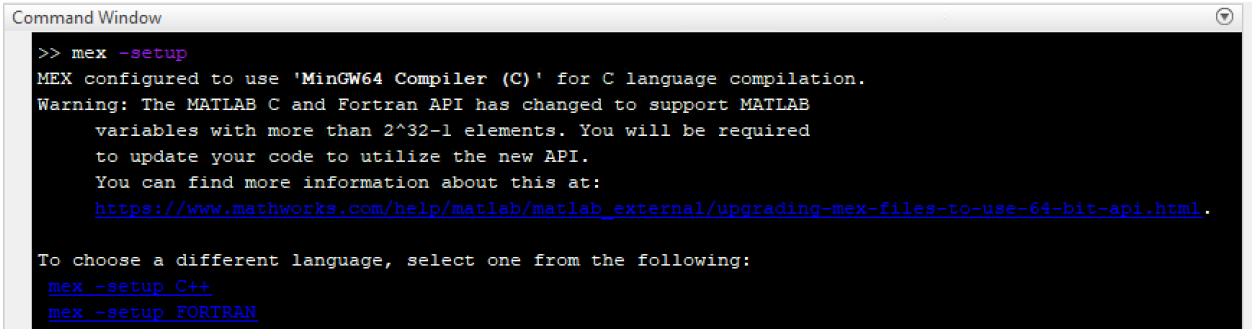


Figure 3: Message display on the command window when a mex C/C++ compiler is stalled successfully.

- Download: TDM-gcc compiler from: http://tdm-gcc.tdragon.net/
- Install TDM-gcc compiler on your computer. For example, I install it at 'C:\TDM-GCC-64' [1].
- Restart MATLAB and input these codes on the command window:
  `setenv('MW_MINGW64_LOC','YourTDMGCCPath'); mex -setup;`
  to check whether "mex" is set up successfully or not. Don't forget to replace the above `'YourTDMGCCPath'` using your own installation location of TDM-gcc package in the last step. For example, if it's installed at 'C:\TDM-GCC-64', please replace previous `'YourTDMGCCPath'` with 'C:\TDM-GCC-64'. If a mex C/C++ compiler is installed successfully, a message similar to (Fig. 3) will display on the MATLAB command window.

3.3 Execute code Section 1

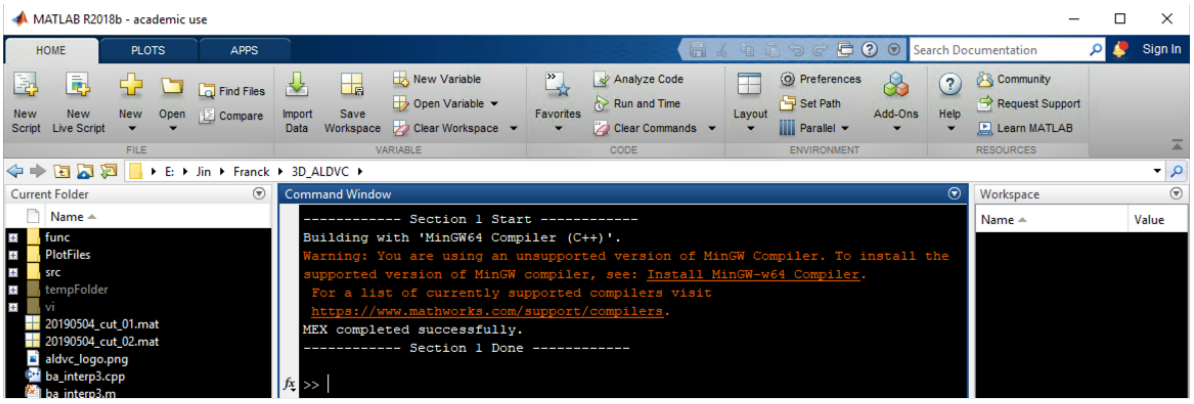
Once a mex C/C++ compiler is installed, we can execute main_ALDIC.m code Section 1 and a successful message will display on the MATLAB command window, see Fig. 4.

4 Code Section 2: Load images and set up DIC parameters & mesh

This section is to load DIC reference and deformed images and set up DIC parameters. First please put your images on the MATLAB working path. After executing this section, all the ALDIC associated parameters will be stored in the workspace in "DICpara". All the mesh properties will be stored in "DICmesh" after executing code Section 3. Here we make a brief summary of both these two data structures in Table 1 and Table 2.

[1] In practice, we find that this TDM-gcc compiler only works if installed on the first level main disks, such as 'C:\', 'D:\', 'E:\', etc.

(a)



(b)

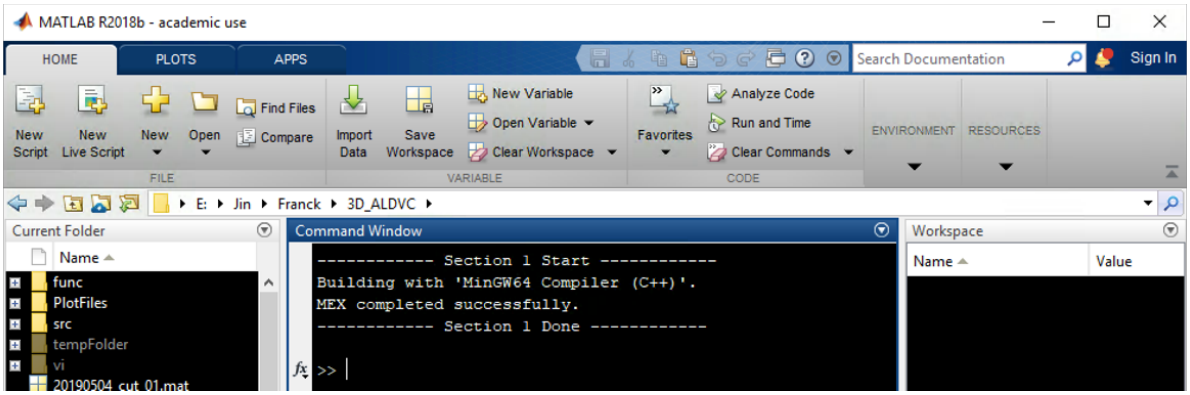


Figure 4: ALDIC code Section 1 is executed successfully and a message similar to (a) or (b) will display on the command window.

Table 1: Summary of DIC parameters in “DICpara” structure

DVCpara variable	DVC parameter	Description and comments
winsize	Subset window size [winsize_x, winsize_y]	Local subset size in ALDIC Subproblem 1
winstepsize	Subset window step [winstepsize_x, winstepsize_y,	The distance between neighboring local subsets, or the finite element size in ALDIC Subproblem 2.
gridRange	DIC region of interest (ROI)	ROI can be defined by clicking top-left and bottom-right corner points.
Subpb2-FDOrFEM	0-’FD’: Finite difference method; 1-’FEM’: finite element method	Both finite different and finite element methods are implemented to solve ALDIC Subproblem 2.
ClusterNo	Number of threads to perform parallel computing	ALDIC Subproblem 1 can be sped up by applying parallel computing.
ImgSize	Image size	Size of 2D DIC images
ImgSeqIncUnit	To decide to perform cumulative or incremental DIC mode	Cumulative DIC is always to compare with the first reference frame; incremental DIC could update the reference frame.
ImgSeqInc-ROIUpdate-OrNot	In incremental mode, ROI can be updated at the same time of updating reference image.	It’s recommended to manually update ROI at the same time of updating reference image for measuring large deformations.
NewFFTSearch	Method to update initial guess to solve an image sequence	Result of last frame can be assigned as the initial guess for the next frame.
interpmethod	Interpolation scheme of volumetric image grayscale at subvoxels	Interpolation scheme can be chosen from {’linear’, ’cubic’(recommended), ’spline’}.
displayIterOrNot	Display convergence details of Subproblem 1 IC-GN iterations	0 or 1: Don’t print or print IC-GN convergence info of each local subset.
Subpb1ICGN-MaxIterNum	Maximum IC-GN iterations to solve Subproblem 1	Subsets fail to converge within ’Subpb1ICGNMaxIterNum’ iteration steps will be marked as bad subsets.
ICGNtol	Tolerance threshold of IC-GN iterations in solving Subproblem 1	E.g., ICGNtol takes value of 10^{-4} px by default.

Table 2: Summary of DIC mesh in “DICmesh” structure

DVCmesh variable	DVC parameter	Description and comments
coordinatesFEM elementsFEM	Coordinates of nodal points in the finite element mesh and their connectivity	Linear 4-node quadrilateral (Q4) elements are used here. However, it can be extended to other type of finite elements with arbitrary shape function.
dirichlet neumann	FE-mesh nodal points at the boundary	Indices of nodal points at ROI borders are assigned with Dirichlet or Neumann boundary conditions.
xy0	Regular FE-mesh nodal grids	Only good for regular Q4 FE-meshes.

4.1 Load DIC images

MATLAB command screen displays these lines to allow user to select their choice to load DIC images, which will be explained in Section 4.1.1-4.1.3.

```
------------ Section 2 Start ------------
Choose method to load images:
0: Select images folder;
1: Use prefix of image names;
2: Manually select images.
Input here:
```

One comment for ALDIC code is that it always manipulate the deformed images and tries to transform them back to the reference image to compute their deformation fields which is based on the *Lagrangian* description. If user wants to track the deformation field in the *Eulerian* description, user can select the reference image as the second image, and select the deformed image as the first image and manipulate the reference image to transform to the current deformed image.

4.1.1 Load DIC images from certain folder

If we select method `0: Select images folder` to load DIC images, you will be asked to select the folder path, and all the images within this folder will be loaded automatically. In the *cumulative* DIC mode, the first frame is set to be the fixed reference image by default, while all the subsequent frames in the image sequence are set to be deformed images whose deformations will be tracked in the Lagrangian description. In the *incremental* DIC mode, the reference image can be updated after every certain number ("DICpara.ImgSeqIncUnit") of frames. After loading DIC images, please jump to Section 4.2.

4.1.2 Load DIC images with prefix of image name

If we select method `1: Use prefix of image names` to load DIC images, user also needs to provide prefix text words of their DIC image frames (and all these images need to be added to the MATLAB path as well). For example, if all the DIC images are named in the format as: "img_0001.tif", "img_0002.tif", $\cdots$, use should input " `img_0*.tif` " on the MATLAB command screen to load all the DIC images started with prefix "img_0" and in the "tif" format. After loading DIC images, please jump to Section 4.2.

```
What is prefix of DIC images? E.g. img_0*.tif.
Input here:
```

4.1.3 Load DIC images manually

If we select method `2: Manually select images` , user needs to load DIC images frame by frame manually, see Fig. 6. ALDIC code sets the first loaded image as reference image and other images as deformed images by default.

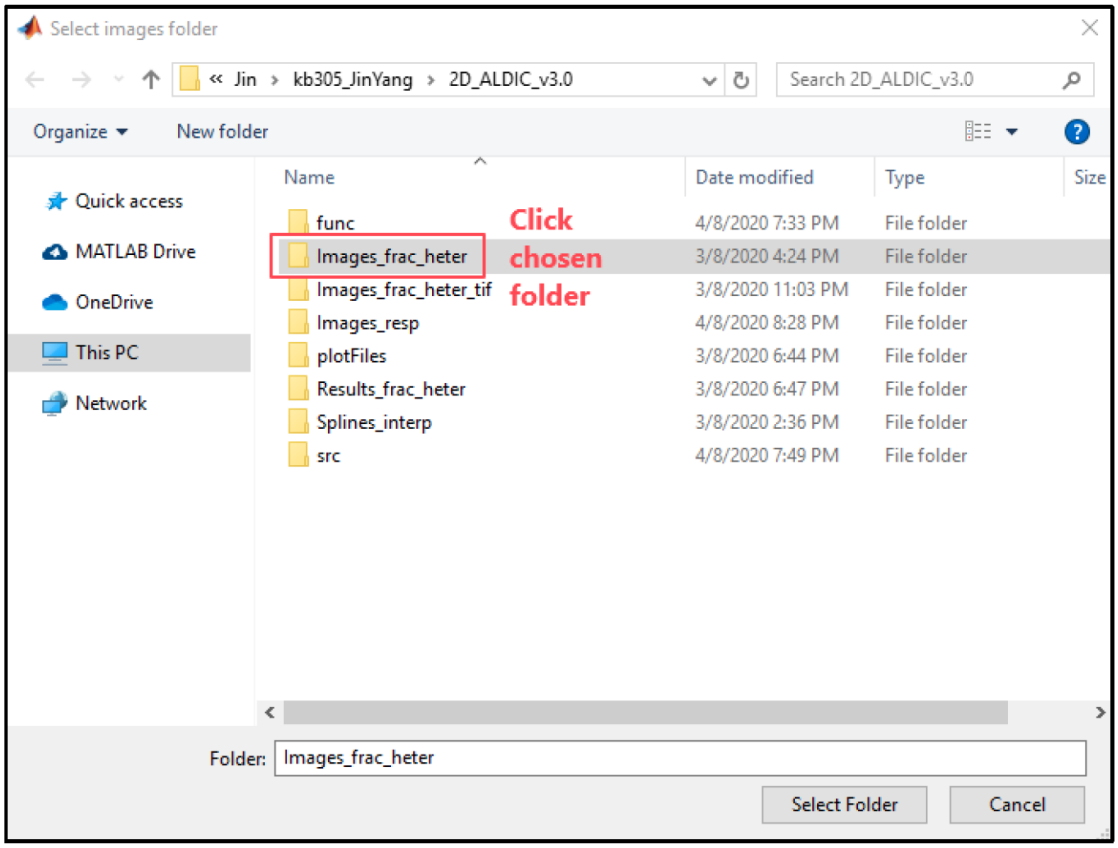


Figure 5: Load DIC iamges by selecting the folder including all DIC images. For example, after clicking chosen folder "Images_frac_heter", all the images in this folder will be loaded automatically.

```
--- Please load first image ---
--- Please load next image ---
Do you want to load more deformed images? (0-yes; 1-no)
```

Input " `0` " if you want to continue uploading images and input " `1` " if you want to stop uploading images. For example, we can select first reference image as "img_0000.tif", and select second image as "img_0570.tif".

4.2 Define region of interest (ROI)

Execute code Section 2, user can click both the top-left and bottom-right corner points on a popped-out image to define DIC region of interest (ROI). On the command window, it displays:

```
--- Define ROI corner points at the top-left and the bottom-right ---
```

Then user first click a left-top point and then click a right-bottom point to define ROI directly on the DIC image, see Fig. 7.

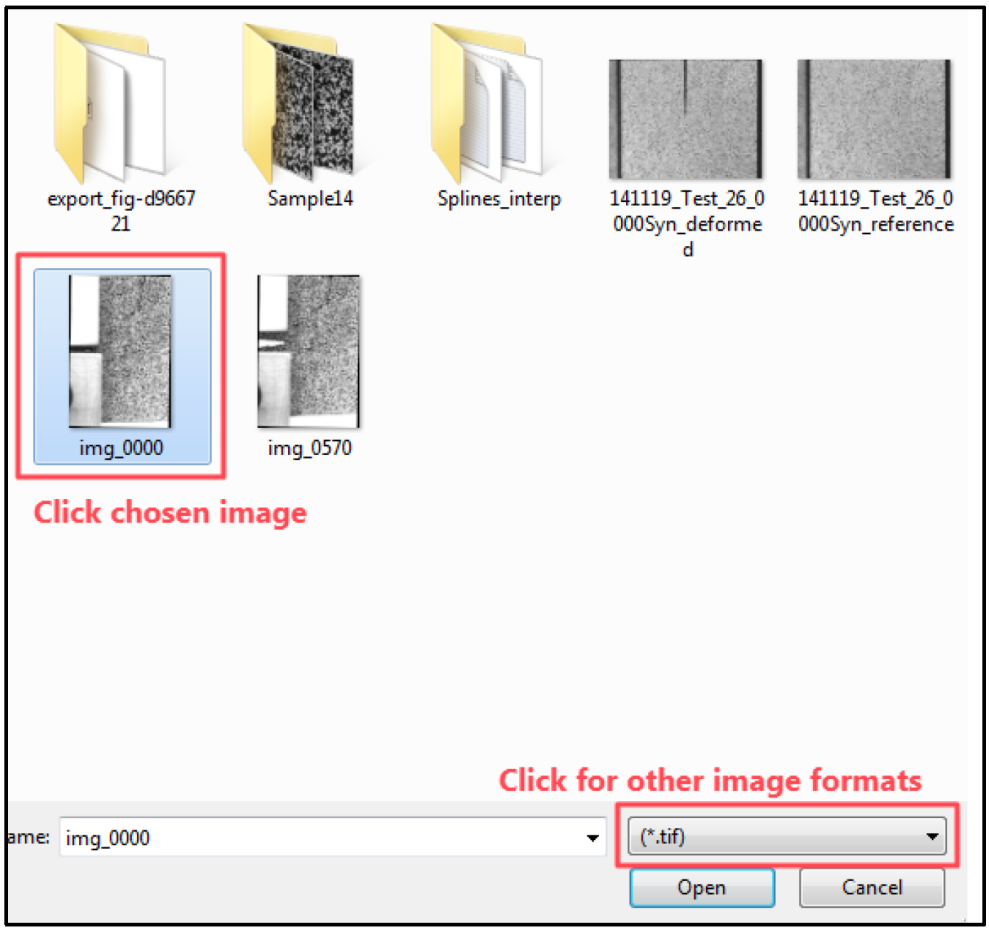


Figure 6: Manually load DIC images by clicking both reference and deformed images.

After clicking both the top-left and bottom-right corner points, the command window screen will display their coordinates in the unit of pixels. E.g., in the image shown in Fig. 7, I define a ROI with top-left and bottom-right corner points are:

```
Coordinates of top-left corner point are (322.786,74.730)
Coordinates of bottom-right corner point are (750.063,1128.439)
```

Comment 1: If the top-left or bottom-right corner point is clicked out of the image, it will be adjusted to the nearest point on the original DIC image borders automatically.

Comment 2: If user prefers to putting in ROI coordinates instead of directly clicking corner points on the DIC image. Please uncomment line `% gridxROIRange = [gridxROIRange1, gridxROIRange2];` `% gridyROIRange = [gridyROIRange1, gridyROIRange2];` and modify values of "gridxROIRange" and "gridyROIRange" there.

Figure 7: Manually click top-left and bottom-right corner points to define DIC region of interest (ROI).

4.3 Set up DIC parameters

Then user will be asked to decide local subset size and the subset step. “Subset size” is the edge length of local subset window; while the “subset step” is the distance between two neighboring subsets, and choice of “subset step” can be independent of choice of subset size.

```
--- What is the subset size? ---
Input here: 20
--- What is the subset step? ---
Input here: 10
```

Subset size is the window size where we use to obtain initial guess through FFT cross correlation method, or through SSD minimization IC-GN iterations in the ALDIC local step. Subset step size is also the finite element mesh size in the ALDIC Subproblem 2 global step. Practically you can choose subset size from 10 px by 10 px to 60 px by 60 px to include 3$\sim$5 features of your DIC images; subset step size can be selected as 0.25$\sim$1 times of subset size. E.g., in the heterogeneous fracture example ”img_0000.tif” and ”img_0570.tif”, we select subset size as `20` px, and subset step is chosen as `10` px.

We also need to choose the solver method for ALDIC Subproblem 2 (global step). For this version of code, we can only deal with uniform grid mesh, and there is almost no difference whether you choose the “Finite difference method” or “Finite element method”. Even you choose “Finite difference method”, there are still finite element mesh generated (cf “DICmesh” in the Matlab workspace) which could help you conduct FEA analysis if you want to combine DIC with other

FEA codes/software afterwards. My personal experience is that finite difference method is faster and a little bit more accurate than finite element method in the ALDIC Subproblem 2 (global step) because of the boundary effects.

```
--- Method to solve ALDIC global step Subproblem 2 ---
    1: Finite difference(Recommended)
    2: Finite element method
Input here: 1
```

We need to set up parallel pools or tell MATLAB we don't want to use parallel pools.

```
--- Set up Parallel pool ---
How many parallel pools to open? (Put in 1 if no parallel computing)
Input here: 4
```

If we don't want to use parallel pools, input `0` or `1` . If we want to use parallel computing, put the number of your parallel pools. E.g., Input `4` .

MATLAB parallel computing environment can be set in the "Home HOME ≫ Environment ≫ Parallel Parallel ≫ Parallel Preferences" (see Fig. 8 and [14] for more information about `parpool`).

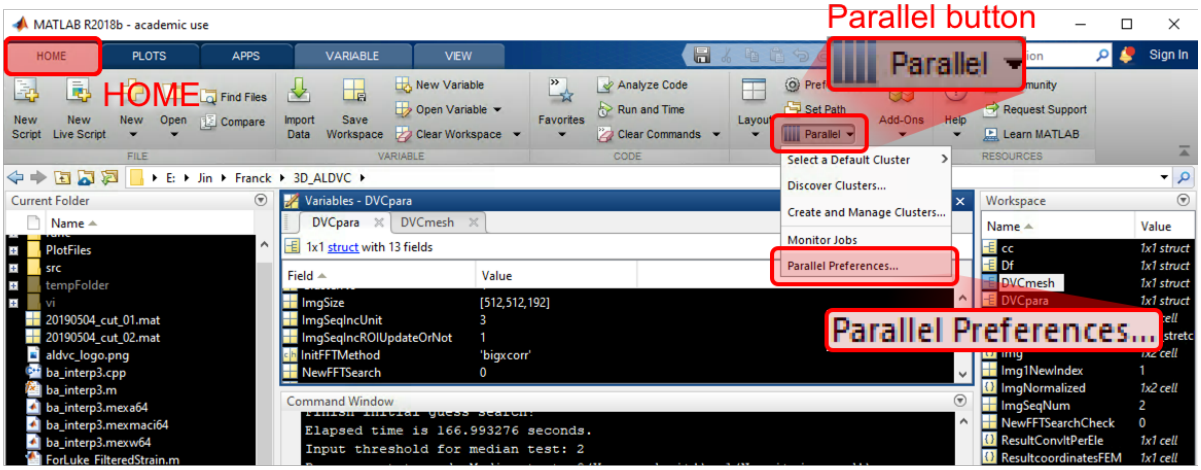
4.4 Additional parameters setup when dealing with image sequence

If we upload an image sequence with more than two frames, as for each new frame, we can choose to use the displacement results in last frame as the initial guess for the new image frame, or we can just redo FFT initial guess for every new frame. This choice depends on how big relative displacements can be between two consecutive frames. Generally, if the relative displacement field between two consecutive frames is smaller than 5-7 voxels, we can use the deformation result of last frame as the initial guess displacement field for the new frame; otherwise it is suggested that we still need to redo the FFT initial guess process.

```
Since we are dealing with an image sequence with multiple frames, for each
    new frame, do we use the result of last frame as an initial guess or
   redo FFT initial guess for every new frame?
0: Use last frame;
1: Redo initial guess.
Input here:
```

When post-process image sequence with more than two frames, user could decide to perform either *cumulative* mode DIC or *incremental* mode DIC. The cumulative mode is the default setup and all the following frames will be compared with the first frame. However, incremental mode is preferred when dealing with extremely large deformations but may lose some accuracy because of the reference image has been updated. We recommend the user to try cumulative mode first, and if cumulative mode doesn't work very well, then try incremental mode. If you choose to use incremental mode, you will further be inquired to input how often you would like to update the reference image:

(a)



(b)

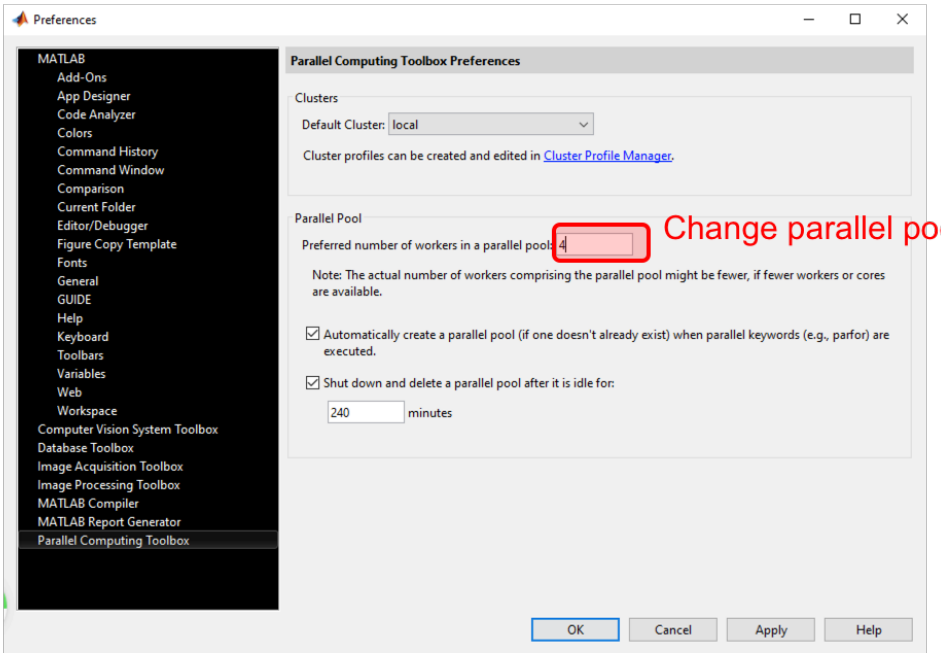


Figure 8: Set up MATLAB parallel preferences. See [14] for more information about `parpool` .

```
--- Choose cumulative or incremental mode ---
     0: Cumulative(By default);
     1: Incremental;
Input here: 1
```

If you choose to use incremental mode, you will further be inquired to input how often you would like to update the reference image:

```
Incremental mode: How many frames to update reference image once?
Input here: 10
```

E.g., I want to update my reference image once every ten frames, so I input: `10` . The minimum number you can input is `1` , which means to update reference image every frame.

Every time the reference image is updated, you can choose to update the region of interest (ROI) at the same time or not. To achieve this, user will be asked as follows:

```
Update ROI at the same time of updating reference image?
    0: Do not update ROI;
    1: Manually(Recommended);
    2: Automatically;
Input here: 1
```

E.g., Input `1` or `2` if you want to update ROI at the same time of updating reference image. Theoretically, this ROI update can be done automatically using the solved deformation of the last frame. However, we find it is most robust to update this ROI manually.

5 Code Section 3: Computing initial guess from FFT-based cross correlation

5.1 FFT-based methods to compute initial guess

In this section, an initial guess of the unknown deformation is computed using fast Fourier transform (FFT) based method to maximize zero normalized cross correlation function C_{ZNCC}.

$$C_{ZNCC} = \frac{\int (f - \mu_f)(g - \mu_g)}{\sigma_f \sigma_g} \tag{1}$$

where $\mathcal{F}(\cdot)$ denotes Fourier transform, $\overline{(\cdot)}$ denotes the complex conjugate, and "$\circ$" denotes the Hadamard product (entry-wise product) and the absolute values are taken entry-wise as well. μ_f, μ_g are average grayscale values of DIC images f and g; σ_f, σ_g are the standard deviation of image grayscale values.

To speed up the above FFT-based optimization process, we apply a multiscale scheme based on a constructed image pyramid. For example, an initial guess for above heterogeneous fracture example is solved in Fig. 9.

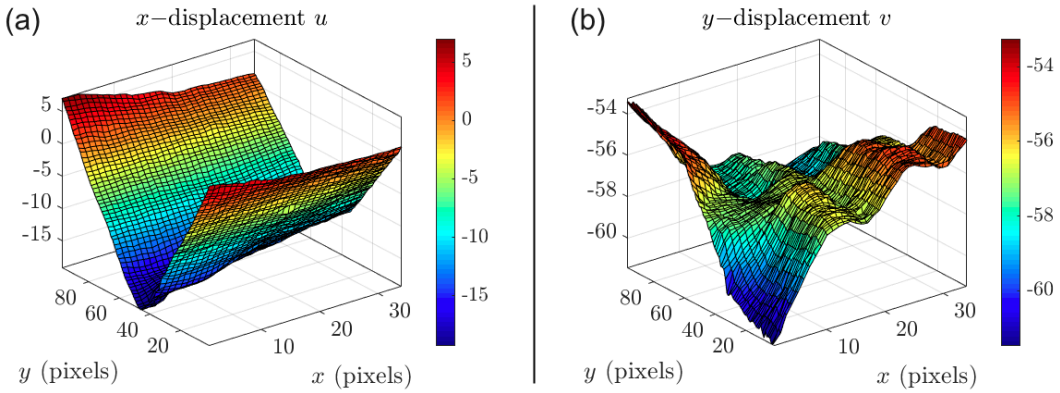


Figure 9: Solved initial guess for displacement fields by FFT-based method to maximize zero normalized cross correlation function.

5.2 Remove noise in computed initial guess

In practice, FFT-based method solved initial guess[2] may have large noise, user can further remove these bad points by applying a median filter, setting a q-factor threshold and setting both upper and lower bounds of displacements. User can also continue to remove bad points by directly clicking them on each image frame and then press " `Enter` " key. However, this step needs some manual work. So this step is **not recommended** until it's very necessary for the overall quality improvement of final ALDIC results. If this noise removal is necessary, user could follow these steps and guide lines will display on MATLAB command window automatically:

```
Do you clear bad points by setting upper/lower bounds once more? (0-yes;
    1-no)
```

If user inputs " `0` ", then he/she will be asked to set the upper and lower bounds of the x- and y-displacements. If user inputs " `1` ", this process will be skipped.

```
% ======= Find bad initial guess points manually by setting bounds =======
What is your upper bound for x-displacement?
What is your lower bound for x-displacement?
What is your upper bound for y-displacement?
What is your lower bound for y-displacement?
Do you clear bad points by setting upper/lower bounds? (0-yes; 1-no)
```

Besides setting upper and lower bounds to remove local bad points, we can continue to remove bad points by clicking them directly.

```
% ======= Find bad initial guess points manually =======
'Do you clear bad points by directly pointing x-disp bad points? (0-yes;
    1-no)';
```

[2]Some designed filters or iterative deformation method (IDM) can further improve the accuracy of initial guess, cf [15, 16, 17, 18]. These are beyond the scope of this code manual. Additionally, accuracy of computed initial guess will further be improved after executing ALDIC ADMM iterations (code Sections 4-6).

```
'Do you clear bad points by directly pointing y-disp bad points? (0-yes;
   1-no)';
```

Directly clicking all the bad points and then press " `Enter` " key.

At the end of this section, a finite element mesh will be generated automatically.

```
--- Finish setting up mesh and assigning initial value! ---
```

Image pixel grayscale value gradients will also be computed very fast using finite difference operator and convolution operations.

```
--- Start to compute image gradients ---
--- Computing image gradients done ---
```

6 Code Section 4: ALDVC Subproblem 1 (first local step)

6.1 Local subset IC-GN solver

In this section, we solve ALDIC ADMM Subproblem 1 (local step) using IC-GN (inverse compositional-Gauss Newton) scheme, where distributed parallel computing has been implemented. Execute this section, IC-GN solver will work and a wait-bar will pop out automatically and allows user to visualize when a program will finish solving ALDIC Subproblem 1, see Fig. 10. If you have very large DIC images, MATLAB parpool may spend several minutes before initializing the parallel computing and transferring all the image dataset. So please don't worry and wait a little bit more if you don't see a waitbar popped out immediately.

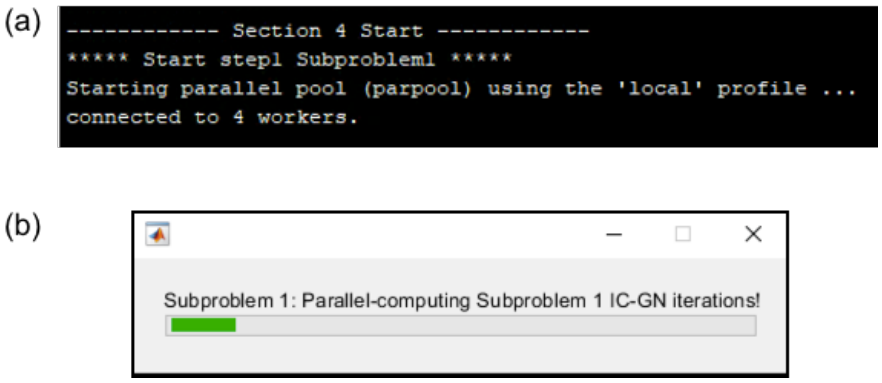


Figure 10: (a) Code section 4 starts and 4 threads are connected. (b) A waitbar pops out and allows user to visualize when a program will finish solving ALDVC Subproblem 1.

```
------------ Section 4 Start ------------
***** Start step1 Subproblem1 *****
Local ICGN bad subsets %:
Elapsed time is xxx seconds.
```

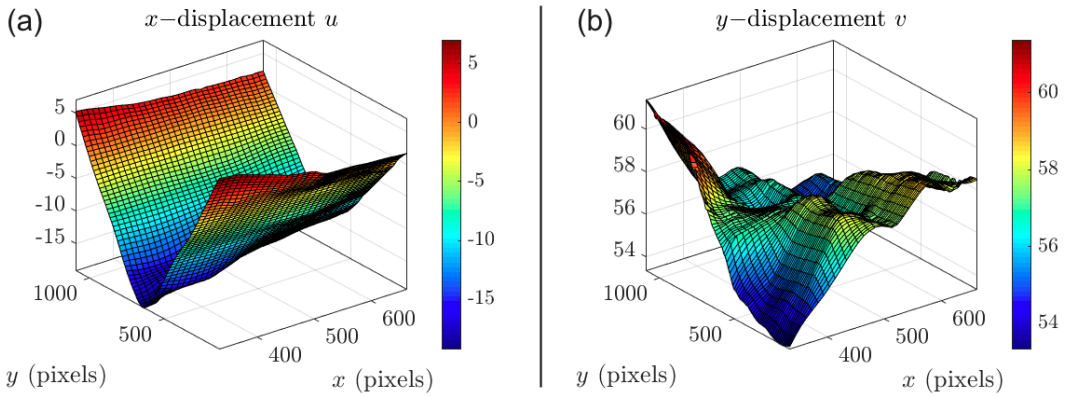


Figure 11: Solved displacements from ALDIC subproblem 1 first local step.

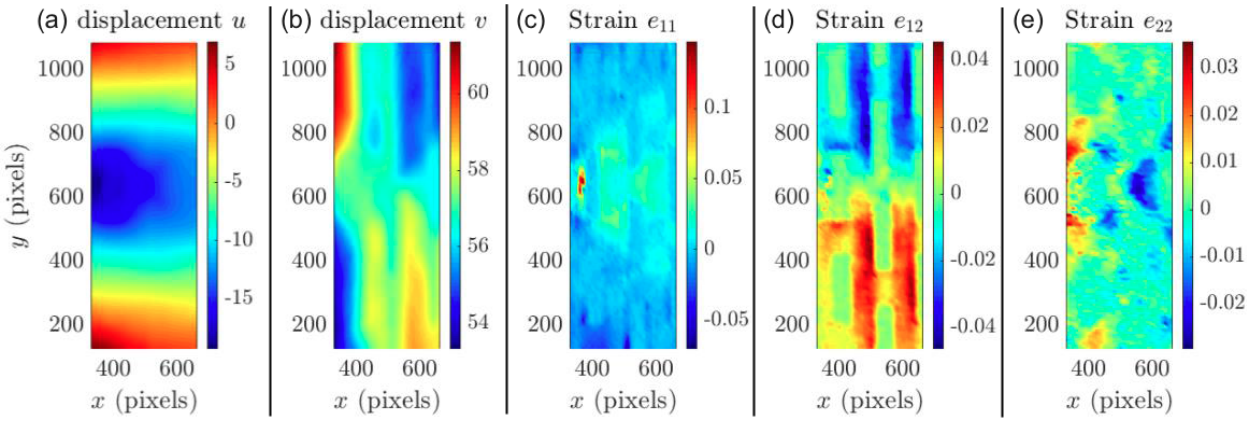


Figure 12: Contour plots of solved displacements and infinitesimal strains from ALDIC subproblem 1 first local step.

6.2 Remove results of IC-GN bad subsets

When this step finishes, a report will display on the command window. Same as Section 5, user can further remove these bad points by applying a median filter, setting a q-factor threshold and setting both upper and lower bounds of displacements. User can also continue to remove bad points by directly clicking them on each image stack and then press " `Enter` " key.

```
--- Start to remove bad local points ---
% ======= Find bad Local IC-GN step points manually by pointing them
    =======
'Do you clear bad points by directly clicking x-disp bad points? (0-yes;
    1-no)';
'Do you clear bad points by directly clicking y-disp bad points? (0-yes;
    1-no)';

% ======= Find bad guess points manually by setting bounds =======
What is your upper bound for x-displacement? % User input
What is your lower bound for x-displacement? % User input
```

```
What is your upper bound for y-displacement? % User input
What is your lower bound for y-displacement? % User input
Do you clear bad points by setting upper/lower bounds? (0-yes; 1-no) %
   Input "0" if you want to redo it.

--- Remove bad points done ---
------------ Section 4 Done ------------
```

Comment: If you see this error message:

```
'Insufficient data for surface estimation.','[u1temp] = gridfit(
   coordinatesFEM (notnanindex,1), coordinatesFEM(notnanindex,2),U(2*
   notnanindex-1), Coordxnodes, Coordynodes,'regularizer','springs');
   u1temp = u1temp';'and 'main_ALDIC (line 106) LocalICGN(U0,DICmesh.
   coordinatesFEM,Df,fNormalized,gNormalized,DICpara,'GaussNewton',tol);'.
```

Or if you see that all the local subsets diverged, in another word, the percentage of local ICGN bad subset is 100%, this usually happens because of the bad DIC parameter choice or bad DIC pattern quality. Here are a few steps that may help fix the poor convergence.

(i) **Check DIC parameter.** In ALDIC code Section 2, each subset is expected to have enough features to track their deformations. With larger subset size ("DICpara.winsize"), the ALDIC ADMM Subproblem 1 will have higher convergence ratio. Typically, we hope there are at least 3∼5 features (e.g. speckle dots, short fibers) within each local subset. Besides subset size, considering both computation speed and accuracy, the distance between neighboring subsets (DICpara.winstepsize) is recommented to be 0.25∼1 times of "DICpara.winsize".

(ii) **Plot and check initial guess** Check the initial guess (around main_ALIDC.m lines 56 64), plot initial guess U0 to see whether this initial guess makes sense or not:
`U0 = Init(u,v,cc.max,DICmesh.x0,DICmesh.y0,0); PlotuvInit;`
Here the newest version of code is to use the multiscale-method to search the initial guess in an adaptive way (line 60, `IntegerSearchMg`). If this initial guess doesn't look great, please replace line (60) with line (58), where you can manually define the fixed size of FFT-search zone, also please make sure the size of search zone (`DICpara.SizeOfFFTSearchRegion`) is larger than the magnitude of displacement vector ($\max(u_x, u_y)$) if you are using non-multiscale " `IntegerSearch` ".

(iii) **Change parameters of IC-GN solver.** For example, increasing `DICpara.Subpb1ICGNMaxIterum` , or decreasing `DICpara.ICGNtol` may help improve IC-GN iteration convergence.

(iv) After checking above (i-iii), user can re-run code Section 4 to see whether most local subsets get convergence or not. User could uncomment `LocalICGN.m: line 35 (ClusterNo=1` or line 58 `ClusterNo>1` to display this info message on the command window (However, this will slow down the code). So please still comment that display line after you fix this issue.

7 Code Section 5: ALDIC Subproblem 2 (first global step)

After solving ALDIC ADMM Subproblem 1 local step, we run Subproblem 2 global step in this section. Both finite difference and finite element methods are implemented. For uniform regular grid meshes, both finite difference and finite element method work very well. For arbitrary mesh, finite element method is much easier to implement and can be easily implemented with adaptive mesh technique (*adapt-ALDIC* code will be available soon).

7.1 Finite difference method

To solve Subproblem using finite difference method, a sparse finite difference operator **D** is generated by `funOperator3` satisfying:

$$\{\mathbf{F}\} = \mathbf{D}\{\mathbf{u}\} \tag{2}$$

where **u** is displacement, and **F** is "deformation gradient tensor minus identity", whose associated components related to node (i) are

$$\begin{aligned} \mathbf{F} &= \left\{ \cdots \quad F_{11}^{(i)} \quad F_{21}^{(i)} \quad F_{12}^{(i)} \quad F_{22}^{(i)} \cdots \right\}^T \\ \mathbf{u} &= \left\{ \cdots \quad u_1^{(i)} \quad u_2^{(i)} \quad \cdots \right\}^T \end{aligned} \tag{3}$$

```
------------ Section 5 Start ------------
***** Start step1 Subproblem2 *****
Assemble finite difference operator D
Elapsed time is xxxxxx seconds.
Finish assembling finite difference operator D
***** Start step1 Subproblem2 *****
Elapsed time is xxxxxx seconds.
------------ Section 5 Done ------------
```

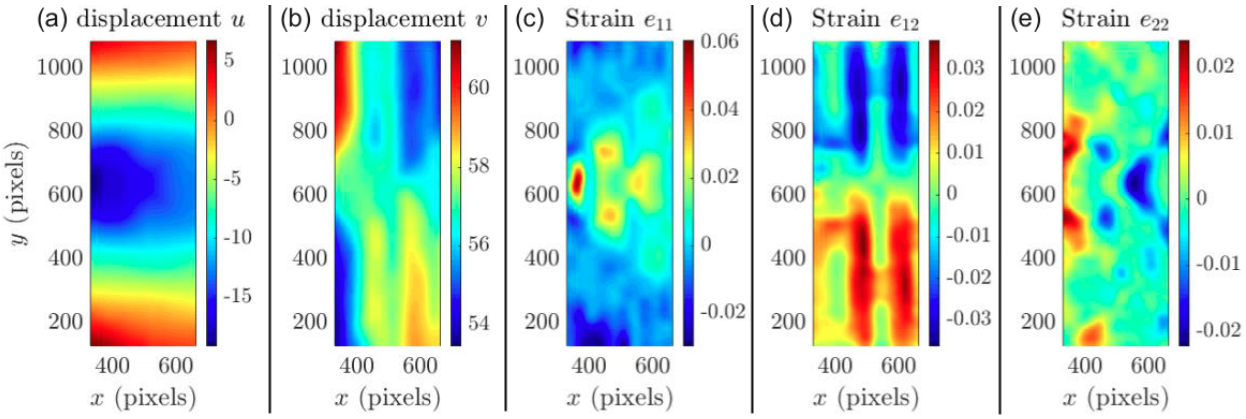


Figure 13: Contour plots of solved displacements and infinitesimal strains from ALDIC subproblem 2 first global step.

7.2 Finite element method

Subproblem 2 can also be solved using finite element method. In this code, we use an 4-node quadrilateral (Q4) uniform finite element, the same element type with our global DIC code (finite element based global DIC code will be available soon).

7.3 Comparison between finite difference and finite element methods in solving Subproblem 2

For uniform regular grid meshes, both finite difference and finite element method solve Subproblem 2 very well and provide accurate results. In practice, finite difference method behaves a little bit better than finite element method in the computation speed and has better accuracy near the ROI boundary. However, finite element method is much easier to implement with an arbitrary mesh, and can be easily combined with adaptive mesh technique.

8 Code Section 6: ALDIC ADMM iterations

Then ALDIC will do alternating direction method of multiplers (ADMM) iterations in this section. The tolerance threshold of ADMM iteration is set to be `1e-4` by default. But this threshold can be manually adjusted by modifying `tol2` in code Section 6. In practice, ALDIC ADMM usually converges within 3$\sim$5 iterations for the update less than `1e-2` px. For example, in the heterogeneous fracture example, ADMM iteration converges at the sixth step, but there is almost no differene between the solution at the third step, see Fig. 14.

```
------------ Section 6 Start -------
***** Start step2 Subproblem1 *****
Local step bad subsets total # is: 0
Elapsed time is 6.078113 seconds.
***** Start step2 Subproblem2 *****
Elapsed time is 0.007632 seconds.
Update local step = 0.018188
Update global step = 0.028851
**********************************

***** Start step3 Subproblem1 *****
Local step bad subsets total # is: 0
Elapsed time is 6.036331 seconds.
***** Start step3 Subproblem2 *****
Elapsed time is 0.007165 seconds.
Update local step = 0.0027562
Update global step = 0.025722
**********************************

***** Start step4 Subproblem1 *****
Local step bad subsets total # is: 0
```

```
Elapsed time is 5.937378 seconds.
***** Start step4 Subproblem2 *****
Elapsed time is 0.007198 seconds.
Update local step = 0.0019051
Update global step = 0.0018707
**********************************

***** Start step5 Subproblem1 *****
Local step bad subsets total # is: 0
Elapsed time is 5.892075 seconds.
***** Start step5 Subproblem2 *****
Elapsed time is 0.007503 seconds.
Update local step = 0.00015808
Update global step = 0.00027234
**********************************

***** Start step6 Subproblem1 *****
Local step bad subsets total # is: 0
Elapsed time is 6.106291 seconds.
***** Start step6 Subproblem2 *****
Elapsed time is 0.007841 seconds.
Update local step = 2.2404e-05
Update global step = 4.8639e-05
**********************************
```

9 Code Section 7: Check convergence

This section is able to check convergence of ALDIC ADMM iterations and delete some temporary variables in the workspace. This section can be skipped if you don't need it.

10 Code Section 8: Compute strains

10.1 Smooth displacement field if needed

Before computing strains, solved displacement fields can be further denoised if necessary. In most cases, ALDIC solved displacement fields are already denoised, so usually there is no need to further smooth solved displacement fields anymore.

```
------------ Section 8 Start ------------
Do you want to smooth displacement? (0-yes; 1-no)1
```

If you put in " `0` ", a Gaussian smooth filter with standard deviation 0.5 will be applied. If a stronger smoothing filter is needed, please edit the Gaussian filter parameters " `DispFilterSize` ", and " `DispFilterStd` ".

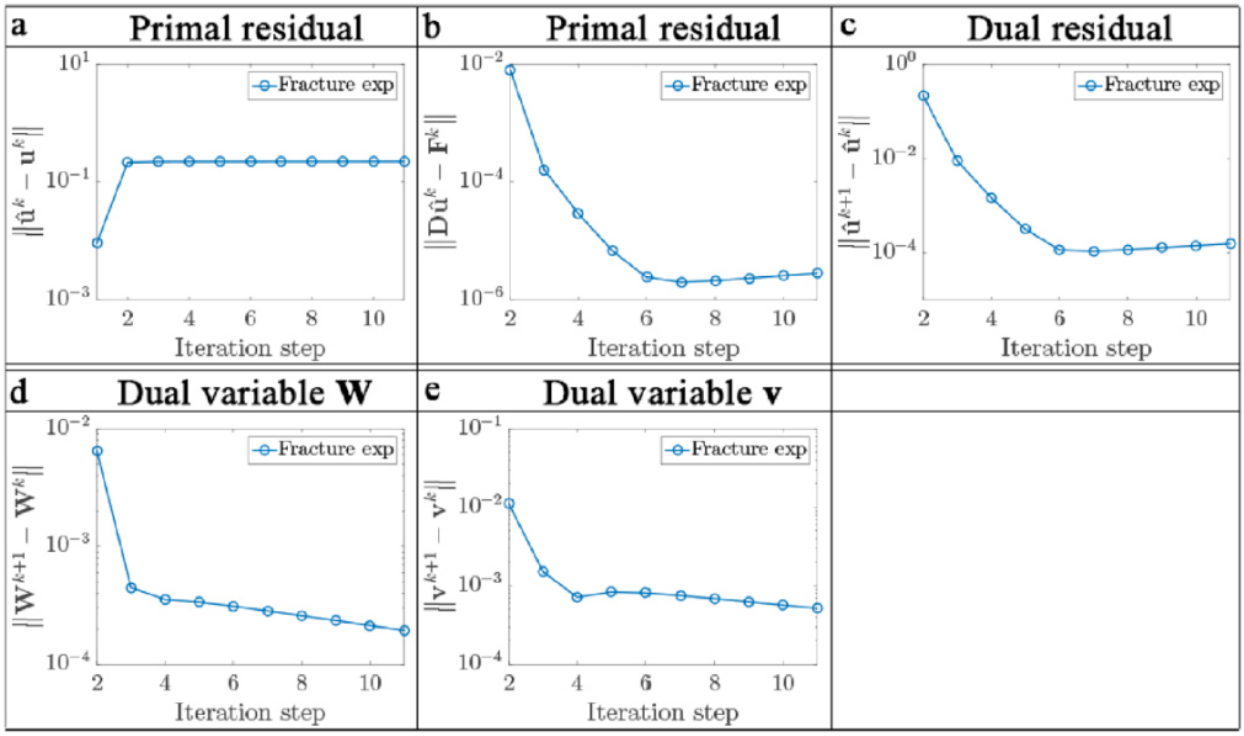


Figure 14: ALDIC ADMM algorithm usually converges after $3 \sim 5$ iterations.

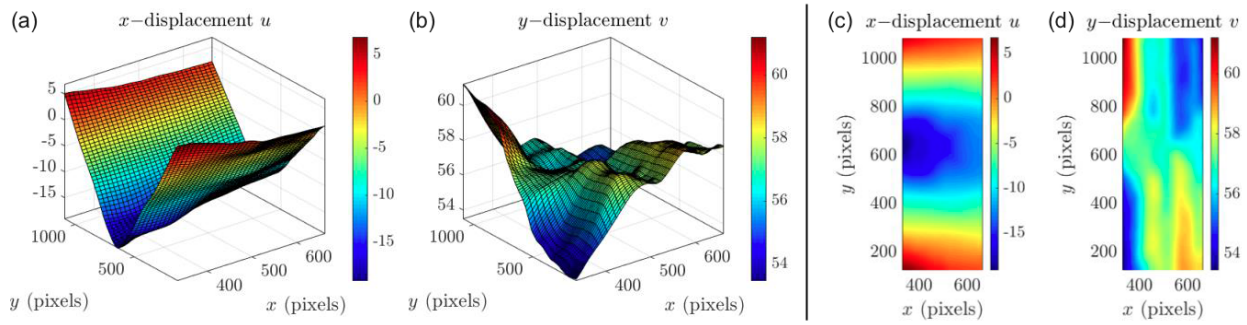


Figure 15: ALDIC solved displacement fields are much already denoised and usually there is no need to apply additional smoothing filters anymore.

10.2 Compute strain field

In ALDIC algorithm, "deformation gradient minus identity" **F** is a direct output besides displacement field. Strain field can also be computed from numerically differentiating solved displacement field. We implement total four methods of computing the strain field.

(0) First method is a **direct output** from ALDIC solved **F** (deformation gradient minus identity);

(1) **Central finite difference** of solved displacement field;

(2) **Plane fitting method** to differentiate solved displacement field. In this method, you will be asked to input half plane width to define the size of the fitted plane;

(3) Strain field can also be computed from **finite element Gauss points**.

```
What method to use to compute strain?
    0: Direct output from ALDVC;
    1: Finite difference(Recommended);
    2: Plane fitting;
    3: Finite element;
Input here: 1
```

If user inputs " `2` ", MATLAB command window will display following lines for continuing to define the plane size:

```
What is your half window size: % Input half window size for plane fitting
```

Three popular types of strains are implemented: infinitesimal strain, Eulerian strain based on deformed configuration, and finite Green-Lagrangian strain.

```
Infinitesimal stran or finite strain?
    0: Infinitesimal stran;
    1: Eulerian strain;
    2: Green-Lagrangian strain;
    3: Others: code by yourself;
Input here: 0
```

E.g., the comparison between different methods to compute strain fields for heterogeneous fracture data set is shown in Fig. 16, where results of this example are stored in " `./results_frac_heter/` ".

10.3 Plot and save results

Users can visualize solved displacement and strain field and plot them overlaid with original DIC images, see Fig. 17. Finally, don't forget to save results for future use.

```
Save figures into different format:
1: jpeg(Choose transparency 0~1)
2: pdf(Choose transparency = 0)
3: Others: Edit codes in ./plotFiles/SaveFigFiles.m
Input here:
```

Current version code, overlaying original DIC image can only be saved in the "jpg" format.

```
Define transparency for overlaying original images:
Input a real number between 0(Only original images)
and 1(Non-transparent deformation results).
Input here(e.g. 0.5):
```

Finally, all the results will be saved in a Matlab matfile, see Table **??**.

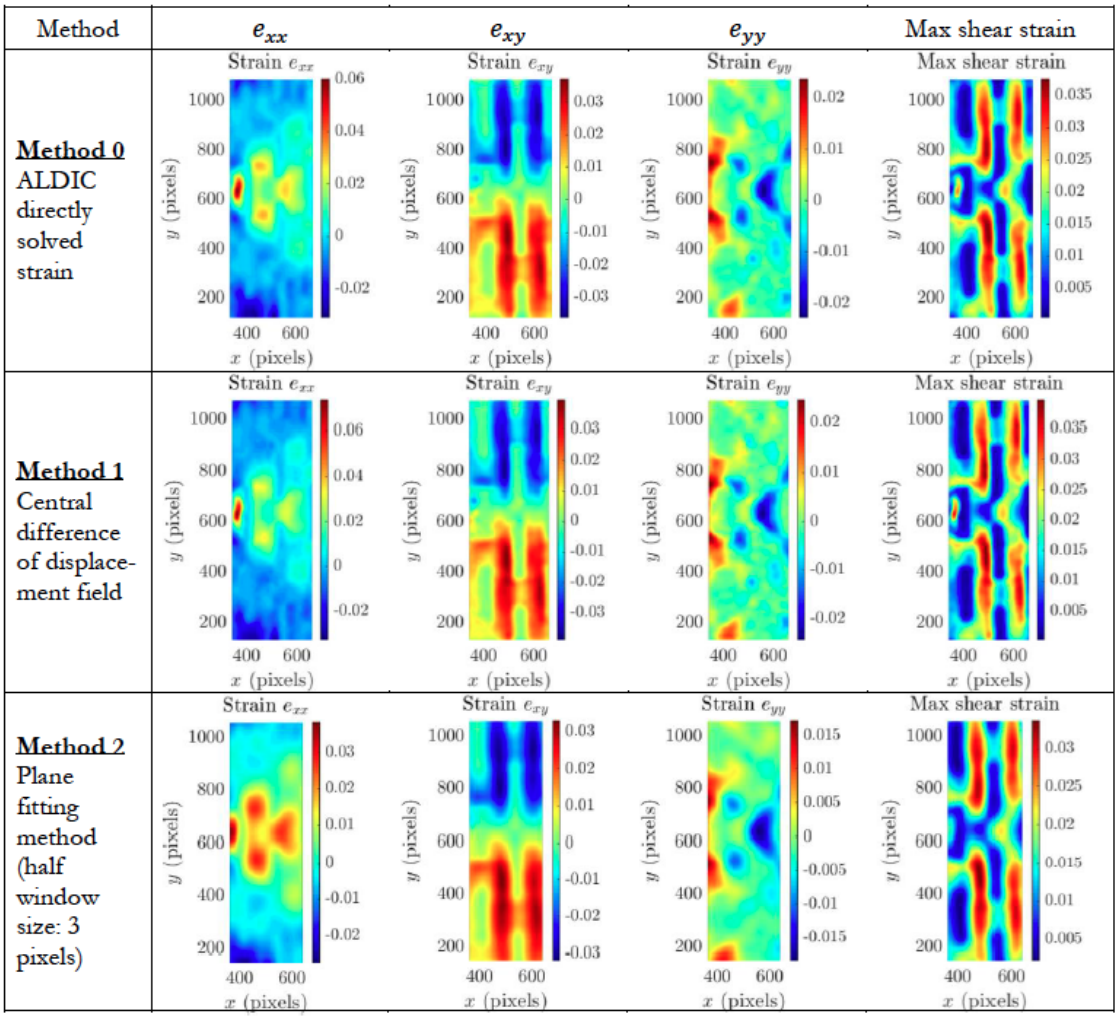


Figure 16: Comparison between different methods to compute strain fields for heterogeneous fracture example.

```
results_name = ['results_ws',num2str(DICpara.winsize(1)),'_st',num2str(
    DICpara.winstepsize(1)),'.mat'];
save(results_name, 'file_name','DICpara','DICmesh','ResultDisp','
    ResultDefGrad','ResultStrain','ResultFEMesh',...
     'ALSub1Time','ALSub2Time', ...
     'ResultcoordinatesFEM','ResultelementsFEM','ResultSizeOfFFTSearchReg')
    ;
```

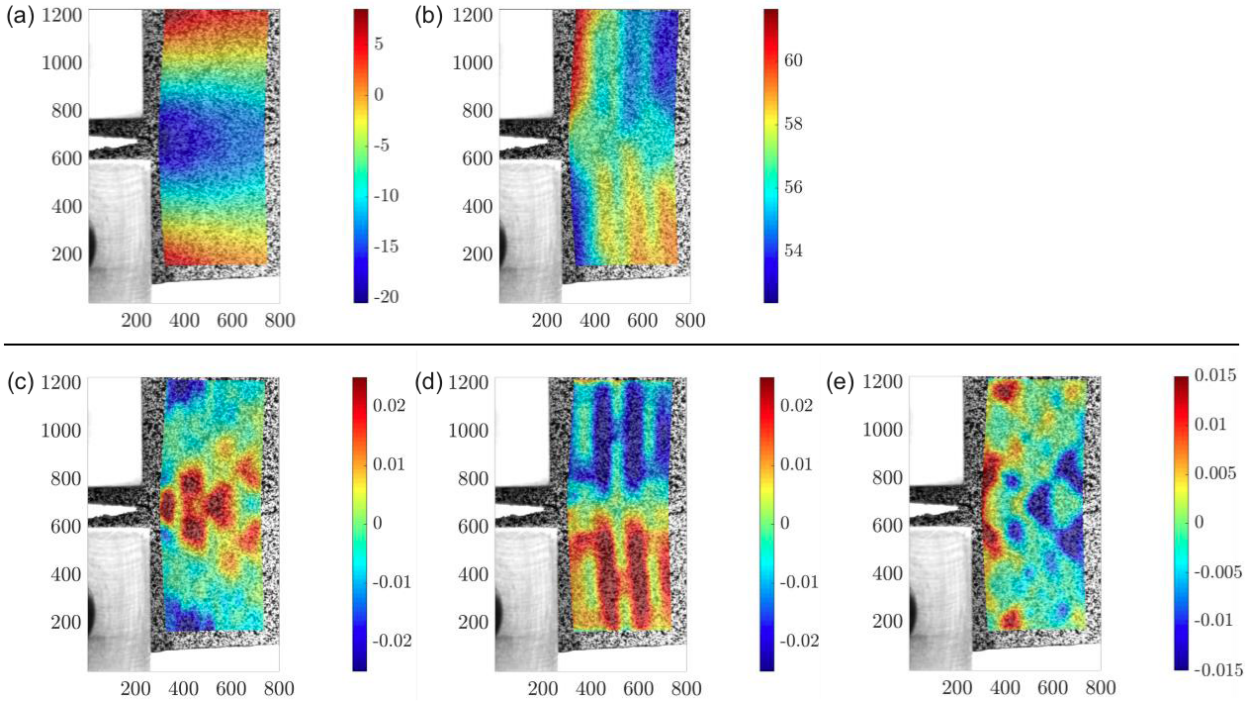


Figure 17: Plotted ALDIC tracked deformation fields with original DIC images as background.

Table 3: Summary of DIC results

Variable	Description
DICpara	ALDIC parameters, see Table 1
DICmesh	ALDIC finite element mesh details, see Table 2
ResultDisp	Solved displacements from ALDIC
ResultDefGrad	Solved "deformation gradient minus identity" **F** from ALDIC
ResultStrain	Solved strains from ALDIC code Section 8
ResultFEMesh	Stored FE-mesh due to reference frame update in incremental mode
ALSub1Time, ALSub2Time	Computation time in solving Subproblems 1 & 2

11 Frequently Asked Question (FAQs)

11.1 About MATLAB mex set up

[1] Where do I install TDM-gcc compiler?

This is a general question for MATLAB users to apply mex C/C++ compilers where more details can be found online, e.g., [12, 13]. User needs to install suitable mex C/C++ compiler based on their OS and MATLAB versions. Based on my experience, mex C/C++ compiler usually is already installed on MAC system. For Windows users, I usually install the TDM-gcc compiler to the first level main disks like "C:, D:, E:", and higher level disks (like "C/FirstLevel/SecondLevel/...") may not work. If you have better suggestions, I appreciate you could let me know.

[2] What if I don't want to use mex compiler or "ba_interp2"? Can I still use ALDIC?

You can still use ALDIC MATLAB code! You need to convert the "ba_interp2" to other MATLAB interpolation functions. This change needs to be done in the "./func/funICGN.m" and "./func/-funICGN_Subpb1.m" to change the "interpmethod". For example, you can replace the function "ba_interp2" with "interp2", where interp2 is the MATLAB built interpolation function. You can also try other interpolation schemes too. Based on my experience, "ba_interp2" runs much faster than the MATLAB built cubic interpolation script. So if "ba_interp2" still can be used, maybe that will be the best solution.

11.2 About MATLAB parallel computing

[3] Where to set up parallel computing preferences?

First, you need to make sure you have already installed parallel computing toolbox. Then all the parallel computing toolbox preferences can be modified by user as shown in Fig. 8. The maximum number of workers in a parallel pool is also limited by the computer hardware.

[4] What if I don't have a parallel computing toolbox?

You can easily install this toolbox. If you don't want to apply parallel computing, ALDIC still runs very fast, and the parameter "DICpara.ClusterNo" needs to be set as "0" or "1".

11.3 About ALDIC algorithm

[5] MATLAB reports error "Undefined function or variable"

Please make sure you have already added all the subfolders ('./src','./plotfiles', ...) on the MATLAB path.

[6] Error "Insufficient data for surface estimation" when using "gridfit"

This problem usually happens when there are too few local subsets get convergence in Subproblem 1 (local step), which breaks down the gridfit function. Could you please check the bad subsets percentage which is expected to display on the command window. For example, if you have almost 100% bad subsets which means all the subsets are not converged in Subpb 1.

Here are three possible steps to help fix this problem. (i) In code Section 2, Check DIC parameters: subset size (DICpara.winsize). In 2D-DIC, each subset is expected to have at least $3\sim5$ features (e.g. speckle dots) in your DIC pattern. Usually with larger subset size, Subpb 1 will have better convergence. But this subset size cannot be too large since it will also decrease the DIC

overall spatial resolution. Theoretically, distance between neighboring subsets can be an arbitrary integer. Considering speed and accuracy, I recommend the winstepsize (DICpara.winstepsize) to be (0.25 $\sim$ 1)* winsize

(ii) In code Section 3, Check the initial guess (around main_ALIDC.m line 56 $\sim$ 64), plot initial guess U0 to see whether this initial guess makes sense or not:

`U0 = Init(u,v,cc.max,DICmesh.x0,DICmesh.y0,0); PlotuvInit;` . The newest version of code is to use the multiscale method (line 60, " `IntegerSearchMg` "), if this initial guess doesn't look great, please replace line (60) with line (58), where you can manually define the size of FFT-search zone.

(iii) After checking (i-ii), please re-run code Section 4. To see whether each local subset obtains convergence or not, you can uncomment (LocalICGN.m: line 35 (ClusterNo=1) or line 58 (ClusterNo>1)) to display this info message on the command window (However, this will slow down the code). If there are still problems or most subsets still get converged. Feel free to report the error message or send a pair of your image samples to (aldicdvc@gmail.com).

[7] How to compute other types of strains?

(i) What ALDIC directly solved are displacements ($\mathbf{U} = \mathbf{x}(\mathbf{X}) - \mathbf{X}$) and deformation gradients ($\mathbf{F} = \nabla_{\nabla \mathbf{X}}\mathbf{U} = \nabla_{\mathbf{X}}\mathbf{x} - \mathbf{I}$), where $\mathbf{x}$ and $\mathbf{X}$ are coordinates in deformed and reference configurations. $\mathbf{U}$ is stored in `ResultDisp` , and $\mathbf{F}$ is stored in `ResultDefGrad` .

In 2D DIC, the length of $\mathbf{U}$ vector is two times of row # of coordinates (in ResultFEMesh1.coordinatesFEM), and the $\mathbf{U}$ vector is assembled by the nodal displacements $[u^1, v^1, u^2, v^2, \cdots, u^N, v^N]^T$. The length of $\mathbf{F}$ vector is four times of row # of coordinates and assembled in the order of $[F_{11}^1, F_{21}^1, F_{12}^1, F_{22}^1, F_{11}^2, F_{21}^2, F_{12}^2, F_{22}^2, \cdots, F_{11}^N, F_{21}^N, F_{12}^N, F_{22}^N]^T$.

(ii) With these solved displacements $\mathbf{U}$ and deformation gradients $\mathbf{F}$, we compute strains in code Section 8. For example, if strains are small (¡5%), infinitesimal strain is a good option, where $e_{xx} = F_{11}$, $e_{yy} = F_{22}$, $e_{xy} = 0.5(F_{12} + F_{21})$. For engineering shear strains, $E_{xx} = F_{11}$, $E_{yy} = F_{22}$, $E_{xy} = (F_{12} + F_{21})$. With computed $\mathbf{F}$ or `FStraintemp` , other types of finite strain measurements can be easily computed based on user's choice.

(iii) Comment about difference between $\mathbf{F}$ and `FStraintemp` . In code Section 8, you will be also be asked:

```
What method to use to compute strain?
0: Direct output from ALDIC;
1: Finite difference(Recommended);
2: Plane fitting;
3: Finite element;
```

If you choose method 0, `FStraintemp` is exactly the same with computed $\mathbf{F}$; If you choose method 1-3, the `FStraintemp` is re-computed from the computed u. The size of `FStraintemp` is cropped by "Rad" since the strains near the edges are less accurate. The coordinates of `FStraintemp` are: x0(1+Rad:M-Rad,1+Rad:N-Rad), y0(1+Rad:M-Rad,1+Rad:N-Rad).

Acknowledgements

I am grateful to Dr. Louisa Avellar for sharing her unpublished images of fracture. I also want to thank Prof. Can C. Aydıner, Dr. Alex K. Landauer and Jialiang Tao, and all the ALDIC users for lots of helpful feedback and discussions. I also acknowledge the support of the US Air Force Office of Scientific Research through the MURI grant 'Managing the Mosaic of Microstructure' (FA9550-12-1-0458)..

References

[1] J Yang and K Bhattacharya. Augmented Lagrangian Digital Image Correlation. *Experimental Mechanics*, 59:187–205, 2019.

[2] *2D ALDIC code*. https://www.mathworks.com/matlabcentral/fileexchange/70499-augmented-lagrangian-digital-image-correlation-and-tracking.

[3] S Boyd, N Parikh, E Chu, B Peleato, and J Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Machine Learning*, 3:1–122, 2010.

[4] https://www.researchgate.net/publication/329456141_Augmented_Lagrangian_Digital_Image_Correlation.

[5] J Yang and K Bhattacharya. Combining image compression with digital image correlation. *Experimental Mechanics*, 59:629–642, 2019.

[6] J Yang and K Bhattacharya. Fast adaptive global digital image correlation. In L Lamberti, MT Lin, C Furlong, C Sciammarella, PL. Reu, and MA Sutton, editors, *Advancement of Optical Methods & Digital Image Correlation in Experimental Mechanics, Volume 3*, pages 69–73. Springer, 2019.

[7] J Yang, L Hazlett, A.K. Landauer, and C. Franck. Augmented Lagrangian Digital Volume Correlation. *Experimental Mechanics*, 2020.

[8] https://www.researchgate.net/publication/343676441_Augmented_Lagrangian_Digital_Volume_Correlation.

[9] *ALDVC code*. https://www.mathworks.com/matlabcentral/fileexchange/77019-augmented-lagrangian-digital-volume-correlation-aldvc.

[10] Augmented Lagrangian Digital Volume Correlation (ALDVC) Code Manual.

[11] *2D Image Interpolation with ba_interp2*. https://www.mathworks.com/matlabcentral/fileexchange/20342-image-interpolation-ba_interp2.

[12] *MATLAB Support for MinGW-w64 C/C++ Compiler*. https://www.mathworks.com/matlabcentral/fileexchange/52848-matlab-support-for-mingw-w64-c-c-compiler.

[13] *MathWorks: MinGW-w64 Compiler*. https://www.mathworks.com/help/matlab/matlab_external/install-mingw-support-package.html.

[14] *MathWorks Help Center: parpool*. https://www.mathworks.com/help/distcomp/parpool.html.

[15] E Bar-Kochba, J Toyjanova, E Andrews, K-S Kim, and C Franck. A fast iterative digital volume correlation algorithm for large deformations. *Experimental Mechanics*, 55:261–274, 2015.

[16] AK Landauer, M Patel, DL Henann, and C Franck. A q-factor-based digital image correlation algorithm (qDIC) for resolving finite deformations with degenerate speckle patterns. *Experimental Mechanics*, 58:815–830, 2018.

[17] *FIDVC code*. https://github.com/FranckLab/FIDVC.

[18] *qFIDVC code*. https://github.com/FranckLab/qFIDVC.