Ncorr

Instruction Manual

Version 1.2.2
6/13/2017

Justin Blaber (jblaber3@gatech.edu)
Antonia Antoniou (antonia.antoniou@me.gatech.edu)
Georgia Institute of Technology

Table of Contents

1.1 - Installation Requirements

Version Requirements:

- *Required:* R2009a+
- Recommended: R2015a+

NOTE: Ncorr was developed on MATLAB R2009a and has not been tested on prior versions.

Toolbox Requirements:

- *Required:* Image Processing Toolbox
- *Required:* Statistics Toolbox

Operating System Requirements:

- Recommended: Windows or Linux

NOTE: Ncorr was developed on Windows 7. It has also been tested on Ubuntu 14.04LTS. Ncorr has not been tested on Mac OS.

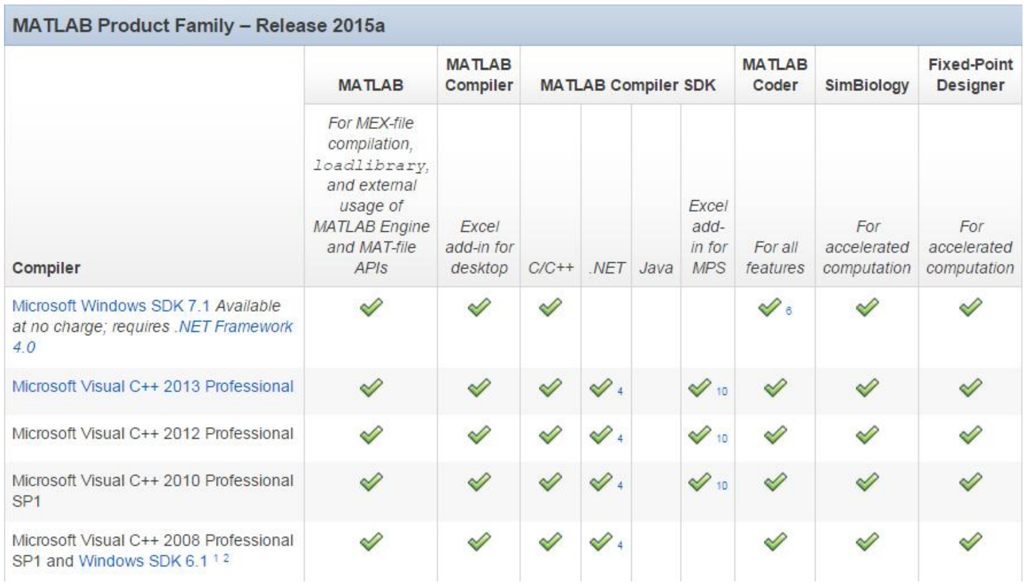
MEX Compiler Requirements:

- Recommended: Visual Studio 2008+ or GCC 4.2+

NOTE: Ncorr utilizes the STL library, so any compiler used must support STL. Ncorr also has an option to use OpenMP for users with multicore processors. If the use of multithreading is desired, then the C++ compiler must have support for OpenMP (do a google search if you aren't sure). If the use of multithreading isn't needed, then any supported compiler which supports the STL library should work.

1.2 - MEX Setup

First, make sure a supported C++ compiler has been installed on your computer. Find the version of your Matlab (i.e. R2015a) and then determine which C++ compilers are supported. For example, R2015a has the following supported compilers:

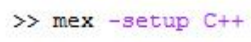


MATLAB Product Family – Release 2015a									
	MATLAB	MATLAB Compiler	MATLAB Compiler SDK				MATLAB Coder	SimBiology	Fixed-Point Designer
Compiler	For MEX-file compilation, `loadlibrary`, and external usage of MATLAB Engine and MAT-file APIs	Excel add-in for desktop	C/C++	.NET	Java	Excel add-in for MPS	For all features	For accelerated computation	For accelerated computation
Microsoft Windows SDK 7.1 *Available at no charge; requires .NET Framework 4.0*	✓	✓	✓				✓ 6	✓	✓
Microsoft Visual C++ 2013 Professional	✓	✓	✓	✓ 4		✓ 10	✓	✓	✓
Microsoft Visual C++ 2012 Professional	✓	✓	✓	✓ 4		✓ 10	✓	✓	✓
Microsoft Visual C++ 2010 Professional SP1	✓	✓	✓	✓ 4		✓ 10	✓	✓	✓
Microsoft Visual C++ 2008 Professional SP1 and Windows SDK 6.1 [1 2]	✓	✓	✓	✓ 4			✓	✓	✓

http://www.mathworks.com/support/compilers/R2015a/index.html

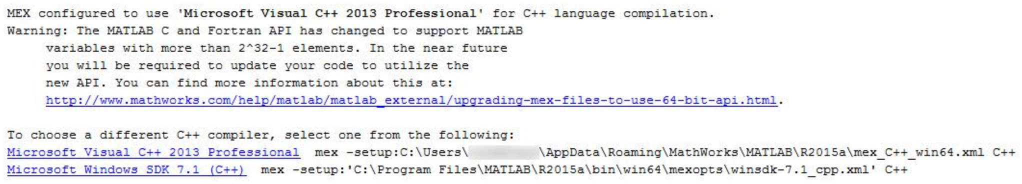
From this list, the Microsoft Windows SDK 7.1 appears to be the only free C++ compiler available for the R2015a release. If you have access to multiple compilers, then use the most recent compiler which has OpenMP support (i.e. if you have access to Microsoft Visual C++ 2013 Professional, then use it). Note that Windows SDK 7.1 does not support OpenMP, so if you are using this package then multithreading cannot be used, although the single threaded version of Ncorr will work.

After downloading a supported compiler, make sure MEX has been set up properly in MATLAB. Type "mex -setup" in the MATLAB terminal as shown below:



```
>> mex -setup C++
```

The output will be something similar to (on Windows using R2015a):



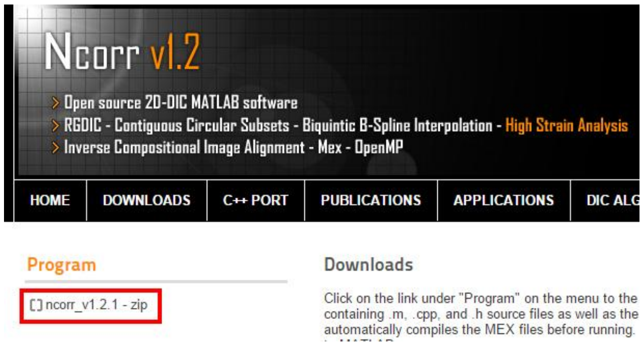
```
MEX configured to use 'Microsoft Visual C++ 2013 Professional' for C++ language compilation.
Warning: The MATLAB C and Fortran API has changed to support MATLAB
     variables with more than 2^32-1 elements. In the near future
     you will be required to update your code to utilize the
     new API. You can find more information about this at:
     http://www.mathworks.com/help/matlab/matlab_external/upgrading-mex-files-to-use-64-bit-api.html.

To choose a different C++ compiler, select one from the following:
Microsoft Visual C++ 2013 Professional  mex -setup:C:\Users\[redacted]\AppData\Roaming\MathWorks\MATLAB\R2015a\mex_C++_win64.xml C++
Microsoft Windows SDK 7.1 (C++)  mex -setup:'C:\Program Files\MATLAB\R2015a\bin\win64\mexopts\winsdk-7.1_cpp.xml' C++
```

This means the compiler is set correctly to Microsoft Visual C++ 2013 Professional. If a compiler isn't set, then follow the instructions in the terminal and set the proper compiler.

1.3 - Automatic Installation

Automatic installation has only been tested on Matlab R2009a and R2015a on Windows using Visual Studio, and R2015a on Ubuntu 14.04LTS using g++. Automatic installation should work with different versions of Matlab on Windows, but for Linux, only R2015a will work. This is because Mathworks updated the MEX interface for R2015a which changed how compiler flags are set in Linux. I've decided to keep things simple and just have automatic installation work best for the newest version of Matlab since this is what most researchers will have access to. If you are using an older version of Matlab on Linux, then perform manual installation (section 1.4). Furthermore, if you are using a different compiler on Windows (i.e. g++ through gnumex), then perform manual installation.

After MEX is set up correctly, get the latest version of Ncorr from the ncorr.com website *downloads* section (as of this manual, the latest version is v1.2.1), as shown below:



Ncorr v1.2

> Open source 2D-DIC MATLAB software
> RGDIC - Contiguous Circular Subsets - Biquintic B-Spline Interpolation - High Strain Analysis
> Inverse Compositional Image Alignment - Mex - OpenMP

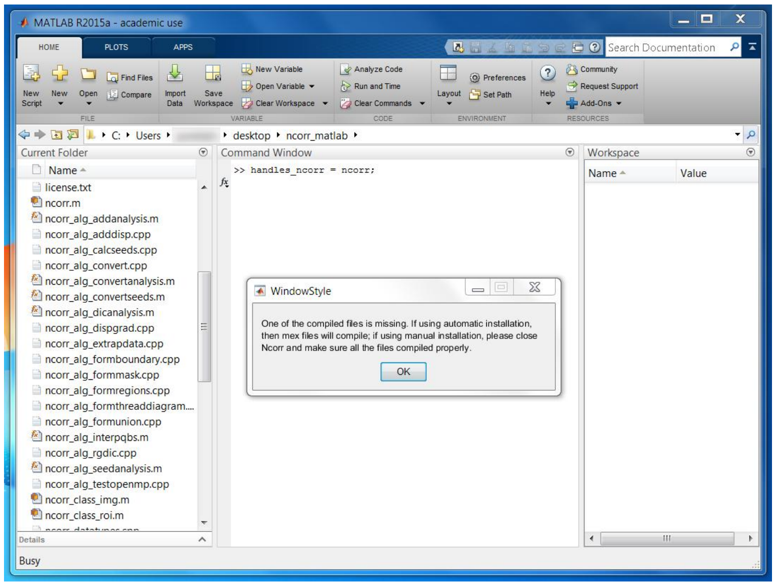
Program

ncorr_v1.2.1 - zip

Downloads

Click on the link under "Program" on the menu to the
containing .m, .cpp, and .h source files as well as the
automatically compiles the MEX files before running.

Then, navigate to the directory where you saved Ncorr. Make sure none of the files have been moved or altered. From there, type "handles_ncorr = ncorr" in the MATLAB terminal as shown below:



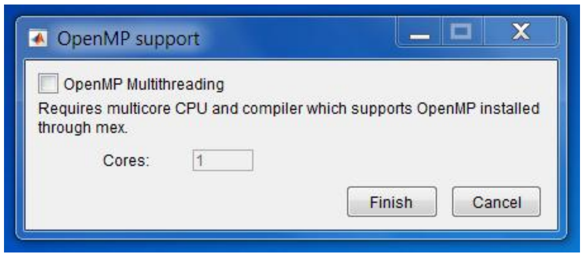
```
>> handles_ncorr = ncorr;
```

WindowStyle

One of the compiled files is missing. If using automatic installation, then mex files will compile; if using manual installation, please close Ncorr and make sure all the files compiled properly.

OK

A dialogue box will appear about the compiled files missing. Since this is a fresh install, the files will not be compiled yet, so click OK to continue. Next, a dialogue box will appear about OpenMP support:

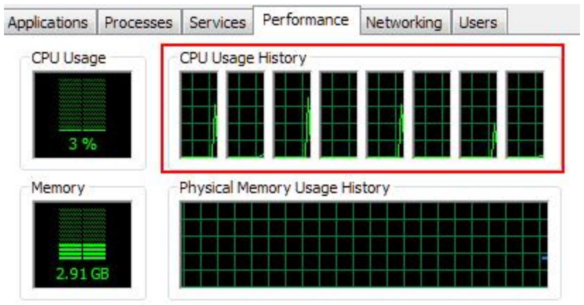

OpenMP support

☐ OpenMP Multithreading

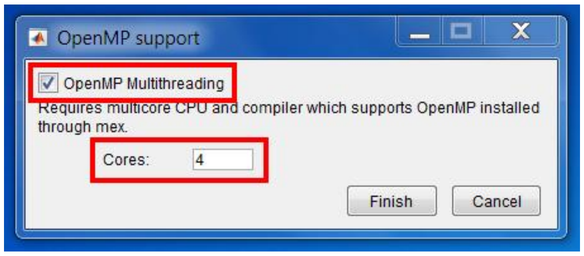
Requires multicore CPU and compiler which supports OpenMP installed through mex.

Cores: 1

Click the checkbox for OpenMP if you have a multicore processor, want multicore support, and have already installed a supported compiler with OpenMP support (section 1.2 – also be sure to google the compiler to see if it supports OpenMP). The number of cores on your system is most easily determined by checking the task manager on Windows (or system processes on Linux) as shown below:



In my case, my computer has 8 logical cores that can be utilized for computation. I only have four physical cores though, so I chose to use 4 cores for computation as shown below:


OpenMP support

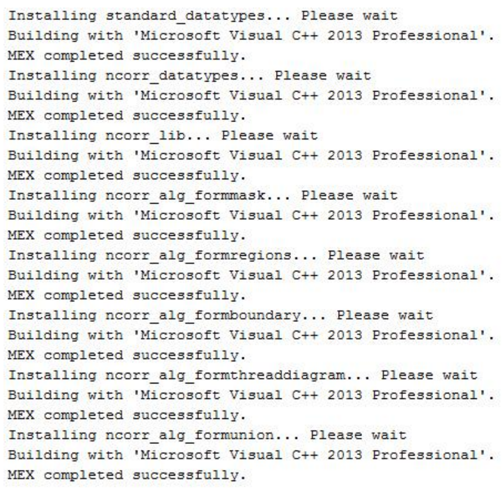
☑ OpenMP Multithreading

Requires multicore CPU and compiler which supports OpenMP installed through mex.

Cores: 4

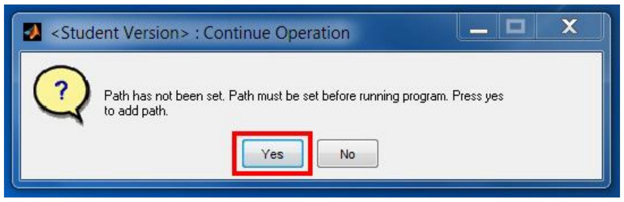
NOTE: If you use all your cores during computation, generally any other applications on the computer will slow down drastically (such as browsing the internet), so if you plan on using your computer during computation, then it might be best to leave out a core or two.

If you choose not to use multithreading, then just leave OpenMP support unchecked and click finish. The files should compile and display in the terminal as shown below:



```
Installing standard_datatypes... Please wait
Building with 'Microsoft Visual C++ 2013 Professional'.
MEX completed successfully.
Installing ncorr_datatypes... Please wait
Building with 'Microsoft Visual C++ 2013 Professional'.
MEX completed successfully.
Installing ncorr_lib... Please wait
Building with 'Microsoft Visual C++ 2013 Professional'.
MEX completed successfully.
Installing ncorr_alg_formmask... Please wait
Building with 'Microsoft Visual C++ 2013 Professional'.
MEX completed successfully.
Installing ncorr_alg_formregions... Please wait
Building with 'Microsoft Visual C++ 2013 Professional'.
MEX completed successfully.
Installing ncorr_alg_formboundary... Please wait
Building with 'Microsoft Visual C++ 2013 Professional'.
MEX completed successfully.
Installing ncorr_alg_formthreaddiagram... Please wait
Building with 'Microsoft Visual C++ 2013 Professional'.
MEX completed successfully.
Installing ncorr_alg_formunion... Please wait
Building with 'Microsoft Visual C++ 2013 Professional'.
MEX completed successfully.
```

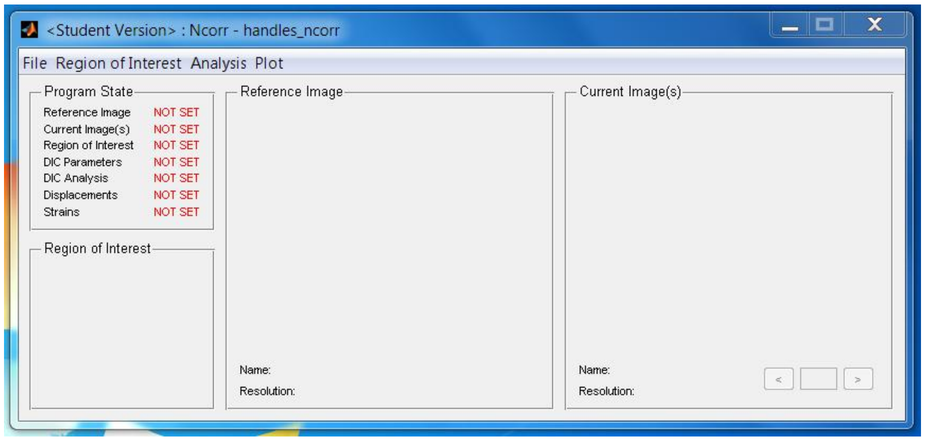
If the files compile successfully (if not, an error message may appear – skip ahead to see what this means), a pop-up for setting the path will appear as shown below:



Path has not been set. Path must be set before running program. Press yes to add path.

The path needs to be set in order to upload images from other directories. This is only a temporary solution and the dialogue box will reappear if MATLAB is closed and reopened. For a more permanent solution, you can set the path manually through: *File > Set Path > Add Folder* from the MATLAB GUI and then clicking save.

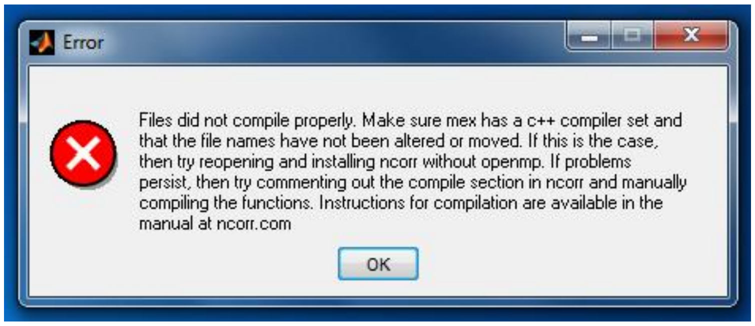
If the files compiled properly, a GUI for Ncorr should appear:


<Student Version> : Ncorr - handles_ncorr
File Region of Interest Analysis Plot
Program State
Reference Image
Current Image(s)
Region of Interest

Program State	
Reference Image	NOT SET
Current Image(s)	NOT SET
Region of Interest	NOT SET
DIC Parameters	NOT SET
DIC Analysis	NOT SET
Displacements	NOT SET
Strains	NOT SET

If this is the case, Ncorr has most likely installed correctly and the installation should be complete. The next time you open MATLAB, you can open Ncorr again by typing "handles_ncorr = ncorr" into the MATLAB terminal and the GUI should appear without having to repeat the installation process. If the installation proceeded correctly, then you can go to the user guides in section 2.

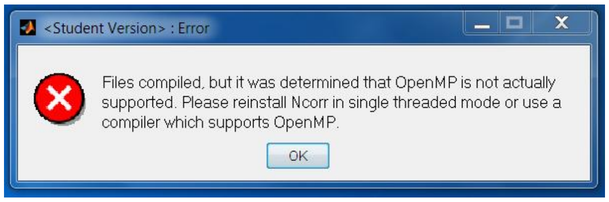
If compilation fails, then the following error message may appear:



Files did not compile properly. Make sure mex has a c++ compiler set and that the file names have not been altered or moved. If this is the case, then try reopening and installing ncorr without openmp. If problems persist, then try commenting out the compile section in ncorr and manually compiling the functions. Instructions for compilation are available in the manual at ncorr.com

In this case, the files did not actually compile properly. If all the above steps have been followed correctly and OpenMP support was enabled, then first try to reinstall Ncorr without OpenMP by clicking “OK”, retyping “handles_ncorr = ncorr” into the terminal, and then leaving the OpenMP support option unchecked. If compilation still fails, then you may need to use a different compiler OR complete the installation manually as described in section 1.4. If you get to this point, feel free to e-mail me at jblaber3@gatech.edu for assistance.

If compilation completes successfully, but it is determined that the compiler doesn’t actually support OpenMP, the following error message will appear:



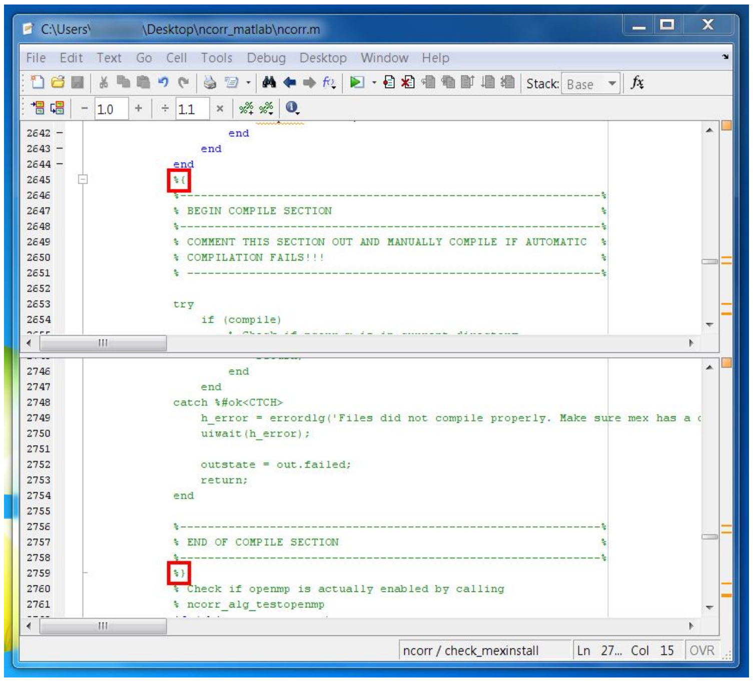
Files compiled, but it was determined that OpenMP is not actually supported. Please reinstall Ncorr in single threaded mode or use a compiler which supports OpenMP.

In this case, you should be able to get Ncorr to work by simply disabling OpenMP support. Just click “OK”, retype “handles_ncorr = ncorr” into the terminal, and leave the OpenMP support option unchecked. If OpenMP support is desired, you need to use a different compiler that supports OpenMP.

1.4 - Manual Installation

These steps show how to perform the manual installation of Ncorr, which involves the compilation of MEX files directly through the terminal, as well as the creation of an "ncorr_installinfo.txt" file which contains information about multithreaded support.

The first step is to comment out the automatic compile section in the **ncorr.m** file. Add "%{" before the compile section and then "%}" after it as shown below:


C:\Users\[redacted]\Desktop\ncorr_matlab\ncorr.m

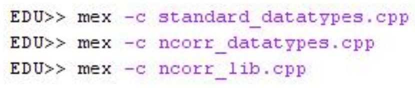
```
                    end
                end
            end
            %{
            %---------------------------------------------------------------%
            % BEGIN COMPILE SECTION                                         %
            %---------------------------------------------------------------%
            % COMMENT THIS SECTION OUT AND MANUALLY COMPILE IF AUTOMATIC    %
            % COMPILATION FAILS!!!                                          %
            % --------------------------------------------------------------%

            try
                if (compile)
```

```
                    end
                end
            catch %#ok<CTCH>
                h_error = errordlg('Files did not compile properly. Make sure mex has a c
                uiwait(h_error);

                outstate = out.failed;
                return;
            end

            %---------------------------------------------------------------%
            % END OF COMPILE SECTION                                        %
            %---------------------------------------------------------------%
            %}
            % Check if openmp is actually enabled by calling
            % ncorr_alg_testopenmp
```

The next step is to manually compile all the necessary libraries and MEX files. At the time of writing this manual, there are three basic library .cpp files, and twelve MEX files which are listed in the table below:

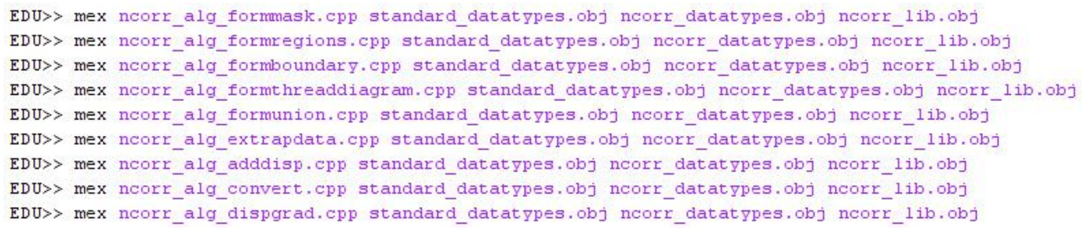
Library Files	standard_datatypes.cpp ncorr_datatypes.cpp ncorr_lib.cpp
MEX Files	ncorr_alg_formmask.cpp ncorr_alg_formregions.cpp ncorr_alg_formboundary.cpp ncorr_alg_formthreaddiagram.cpp ncorr_alg_formunion.cpp ncorr_alg_extrapdata.cpp ncorr_alg_adddisp.cpp ncorr_alg_convert.cpp ncorr_alg_dispgrad.cpp ncorr_alg_testopenmp.cpp ncorr_alg_calcseeds.cpp ncorr_alg_rgdic.cpp

Start by compiling the libraries "standard_datatypes.cpp," "ncorr_datatypes.cpp," and "ncorr_lib.cpp." as object files by using the "-c" flag with MEX as shown below:



```
EDU>> mex -c standard_datatypes.cpp
EDU>> mex -c ncorr_datatypes.cpp
EDU>> mex -c ncorr_lib.cpp
```

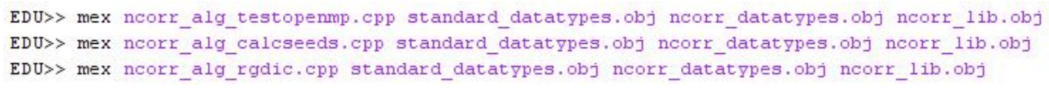
Next, compile the following MEX files with:



```
EDU>> mex ncorr_alg_formmask.cpp standard_datatypes.obj ncorr_datatypes.obj ncorr_lib.obj
EDU>> mex ncorr_alg_formregions.cpp standard_datatypes.obj ncorr_datatypes.obj ncorr_lib.obj
EDU>> mex ncorr_alg_formboundary.cpp standard_datatypes.obj ncorr_datatypes.obj ncorr_lib.obj
EDU>> mex ncorr_alg_formthreaddiagram.cpp standard_datatypes.obj ncorr_datatypes.obj ncorr_lib.obj
EDU>> mex ncorr_alg_formunion.cpp standard_datatypes.obj ncorr_datatypes.obj ncorr_lib.obj
EDU>> mex ncorr_alg_extrapdata.cpp standard_datatypes.obj ncorr_datatypes.obj ncorr_lib.obj
EDU>> mex ncorr_alg_adddisp.cpp standard_datatypes.obj ncorr_datatypes.obj ncorr_lib.obj
EDU>> mex ncorr_alg_convert.cpp standard_datatypes.obj ncorr_datatypes.obj ncorr_lib.obj
EDU>> mex ncorr_alg_dispgrad.cpp standard_datatypes.obj ncorr_datatypes.obj ncorr_lib.obj
```

NOTE: When compiling the MEX files with linux, the standard_datatypes and ncorr_datatypes libraries will be compiled with an ".o" extension instead of an ".obj" extension.

The last three files that need to be compiled are "ncorr_alg_testopenmp.cpp", "ncorr_alg_calcseeds.cpp" and "ncorr_alg_rgdic.cpp." These files utilize OpenMP and thus needs to be compiled with certain compiler flags. If multithreaded support is not desired, then the files can be compiled like the other MEX files in the following way:



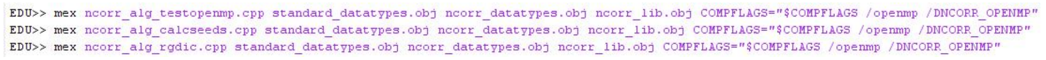
```
EDU>> mex ncorr_alg_testopenmp.cpp standard_datatypes.obj ncorr_datatypes.obj ncorr_lib.obj
EDU>> mex ncorr_alg_calcseeds.cpp standard_datatypes.obj ncorr_datatypes.obj ncorr_lib.obj
EDU>> mex ncorr_alg_rgdic.cpp standard_datatypes.obj ncorr_datatypes.obj ncorr_lib.obj
```

Otherwise, flags need to be passed to the compiler. The required flags are shown in the table below:

OS/Compiler	Flags
Windows with Visual Studio	COMPFLAGS="$COMPFLAGS /openmp /DNCORR_OPENMP"

Linux with GCC	Before R2015a:
	CXXFLAGS="\$CXXFLAGS -fopenmp -DNCORR_OPENMP" CXXLIBS="\$CXXLIBS -lgomp"
	
	R2015a+ (remove the backslash):
	CXXFLAGS="$CXXFLAGS -fopenmp -DNCORR_OPENMP" CXXLIBS="$CXXLIBS -lgomp"

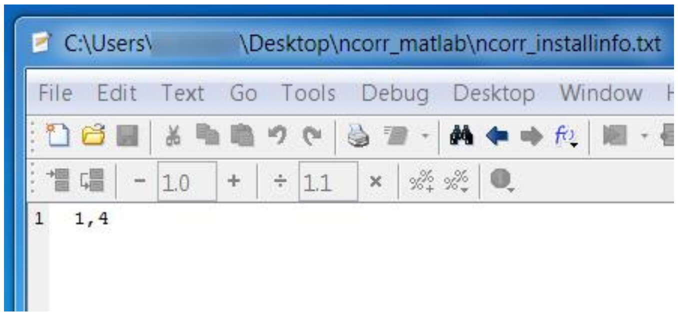
An example compilation for these files with the appropriate flags is shown below for a system with Windows and Visual studio:



```
EDU>> mex ncorr_alg_testopenmp.cpp standard_datatypes.obj ncorr_datatypes.obj ncorr_lib.obj COMPFLAGS="$COMPFLAGS /openmp /DNCORR_OPENMP"
EDU>> mex ncorr_alg_calcseeds.cpp standard_datatypes.obj ncorr_datatypes.obj ncorr_lib.obj COMPFLAGS="$COMPFLAGS /openmp /DNCORR_OPENMP"
EDU>> mex ncorr_alg_rgdic.cpp standard_datatypes.obj ncorr_datatypes.obj ncorr_lib.obj COMPFLAGS="$COMPFLAGS /openmp /DNCORR_OPENMP"
```

The important thing to note here is to make sure you use the correct name and format for the compiler flags, which depend on **both the operating system and the compiler**. If compilation still fails or more assistance is needed then please email me at jblaber3@gatech.edu and include the entire error message as a screen shot.

At this point, all the MEX files should be compiled. The next step is to create a file called "ncorr_installinfo.txt" and then fill it with the following:


C:\Users\[redacted]\Desktop\ncorr_matlab\ncorr_installinfo.txt
File Edit Text Go Tools Debug Desktop Window
1.0
1.1

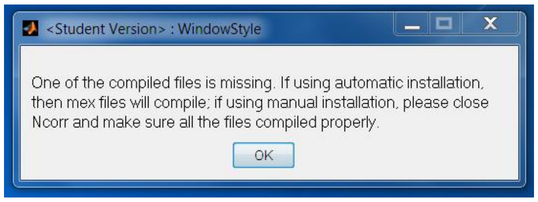
```
1,4
```

The first number represents whether or not OpenMP support exists, and should be either 0 or 1 (0 = no OpenMP support; 1 = OpenMP support). The second number represents how many threads you want, and needs to be **greater than or equal to 1**. For example, the combination can be “0,1” for single threading, or “1,4” as shown above for multithreading with four threads.

NOTE: Make sure to separate the numbers by a comma; the format should be: "#,#".

After the above is complete, type "handles_ncorr = ncorr" into the MATLAB terminal which should bring the Ncorr GUI up. If the main GUI appears directly, then the installation should be complete.

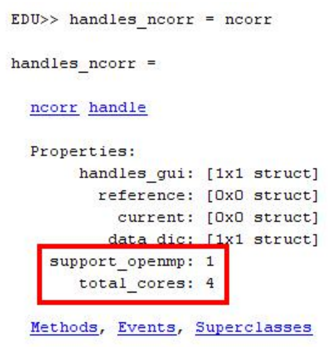
If you forgot to compile a file or forgot to set up the ncorr_installinfo.txt file, then the following will appear:



One of the compiled files is missing. If using automatic installation, then mex files will compile; if using manual installation, please close Ncorr and make sure all the files compiled properly.

If this appears, then go back over this section and make sure you did not forget to compile a file.

If the installation went correctly, then as a double check, make sure the options you specified in the "ncorr_installinfo.txt" file were loaded correctly by typing "handles_ncorr" in the MATLAB terminal to view its properties and compare them with the values you provided in the ncorr_installinfo.txt file:



```
EDU>> handles_ncorr = ncorr

handles_ncorr =

  ncorr handle

  Properties:
       handles_gui: [1x1 struct]
         reference: [0x0 struct]
           current: [0x0 struct]
          data_dic: [1x1 struct]
    support_openmp: 1
       total_cores: 4

  Methods, Events, Superclasses
```

2.1 – Program Flow

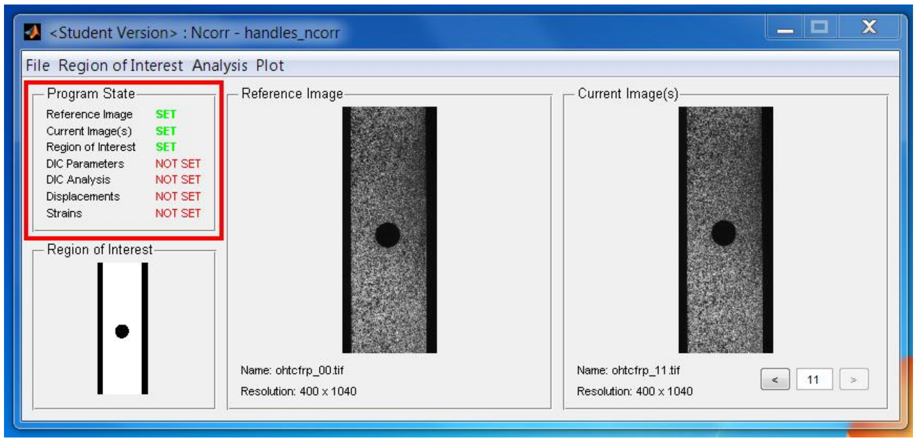
For the general DIC section, I am using the "plate hole" sample from SEM's DIC challenge. The formatted images, along with a ROI, are available off the Ncorr website if the user would like to follow along:

http://ncorr.com/download/sample12.zip

At this point, it is assumed Ncorr has been successfully compiled and installed on the user's computer. The work flow of Ncorr is as follows:

1. Set Reference Image
2. Set Current Image(s)
3. Set Region of Interest (ROI)
   - *Dependencies:* Requires reference image or current image(s) to be set first
4. Set DIC Parameters
   - *Dependencies:* Requires reference image, current image(s), and ROI to be set first
5. DIC Analysis
   - *Dependencies:* Requires DIC parameters to be set first
6. Format Displacements
   - *Dependencies:* Requires DIC Analysis to be run first
7. Calculate Strains
   - *Dependencies:* Requires displacements to be formatted first.

Any of these steps can be altered at any stage, so long as the dependencies have been met. The state of the program is visible at the top left corner of the main Ncorr GUI as shown below:


<Student Version> : Ncorr - handles_ncorr
File Region of Interest Analysis Plot
Region of Interest
Reference Image
Current Image(s)
<
11
>

Program State

Reference Image	SET
Current Image(s)	SET
Region of Interest	SET
DIC Parameters	NOT SET
DIC Analysis	NOT SET
Displacements	NOT SET
Strains	NOT SET

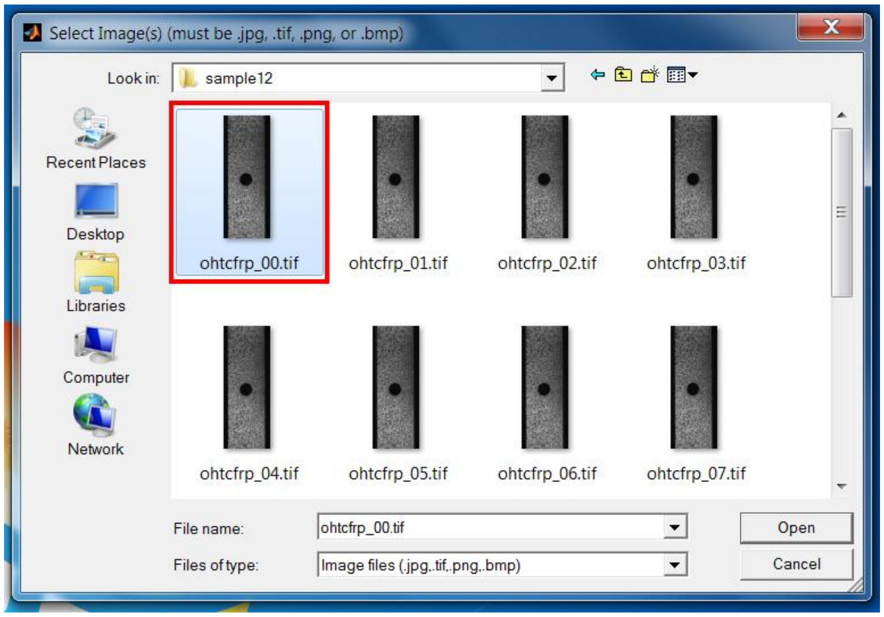
Name: ohtcfrp_00.tif
Resolution: 400 x 1040

Name: ohtcfrp_11.tif
Resolution: 400 x 1040

This lets the user know which stage they are currently on and which should come next. The state is listed in sequential order for convenience. In the example above, the state lets the user know that the reference image, current image, and ROI are loaded and that setting the DIC parameters needs to proceed next.

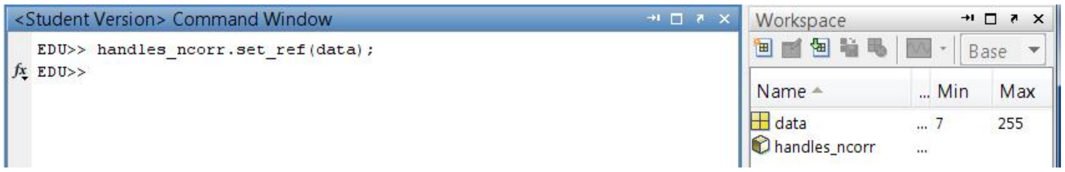
2.2 - Setting the Reference Image

There are two ways to set the reference image. The first way is to use the Ncorr GUI by going to *File > Load Reference Image* which results in the following:



NOTE: The reference image means the initial or first image you take of the sample, before any deformation occurs.

The other way to load a reference image is to do it through the MATLAB terminal. You can set the reference image by typing "handles_ncorr.set_ref(data)," where "data" is a 2D matrix (of type double, uint16, or uint8) containing image grayscale values, as shown below:

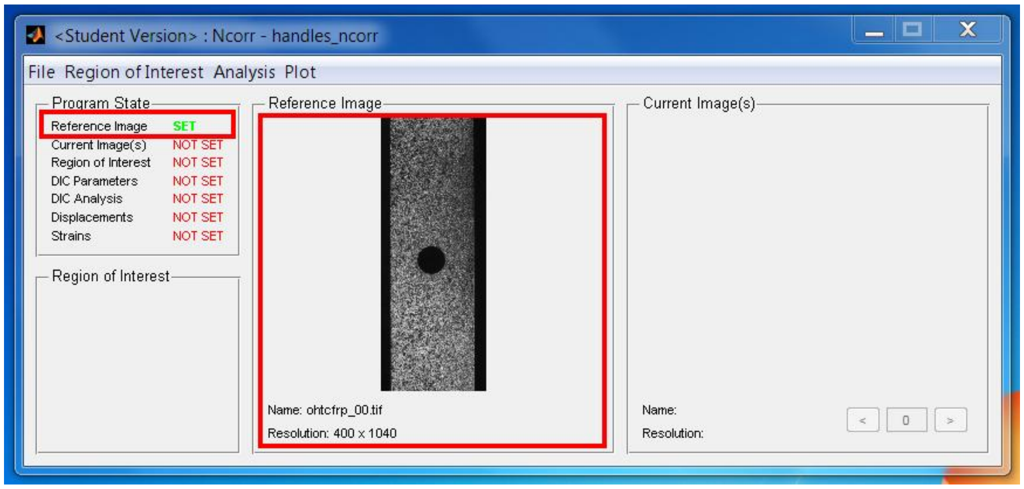


```
EDU>> handles_ncorr.set_ref(data);
EDU>>
```

Name	... Min	Max
data	... 7	255
handles_ncorr	...	

This feature is provided because sometimes the reference image is obtained through a series of images which are averaged together or processed in some way. If this is the case, the reference image will be stored as an array in the base workspace, and it's much more convenient to load it directly than to save it as an image, which can also lead to loss of data through image compression or binning of gray scale values.

Lastly, when the reference image is loaded it should appear as shown below:


<Student Version> : Ncorr - handles_ncorr
File Region of Interest Analysis Plot
Program State
Reference Image
Current Image(s)
Region of Interest
< 0 >

Reference Image	SET
Current Image(s)	NOT SET
Region of Interest	NOT SET
DIC Parameters	NOT SET
DIC Analysis	NOT SET
Displacements	NOT SET
Strains	NOT SET

Name: ohtcfrp_00.tif
Resolution: 400 x 1040

Name:
Resolution:

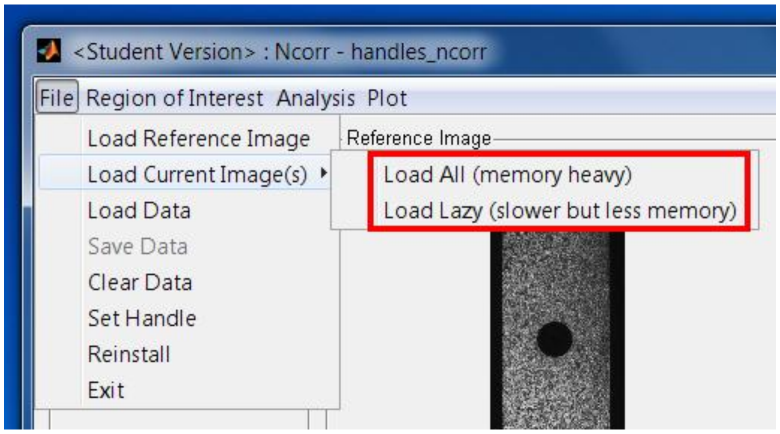
2.3 - Setting the Current Image(s)

First of all, one of the main differences between setting the current image(s) vs the reference image is that more than one image may be loaded for the current (deformed) configuration. Because of this, the program needs a way to order the current image(s) if more than one is loaded. To accomplish this, the images must have the name format shown below:

name_#.ext

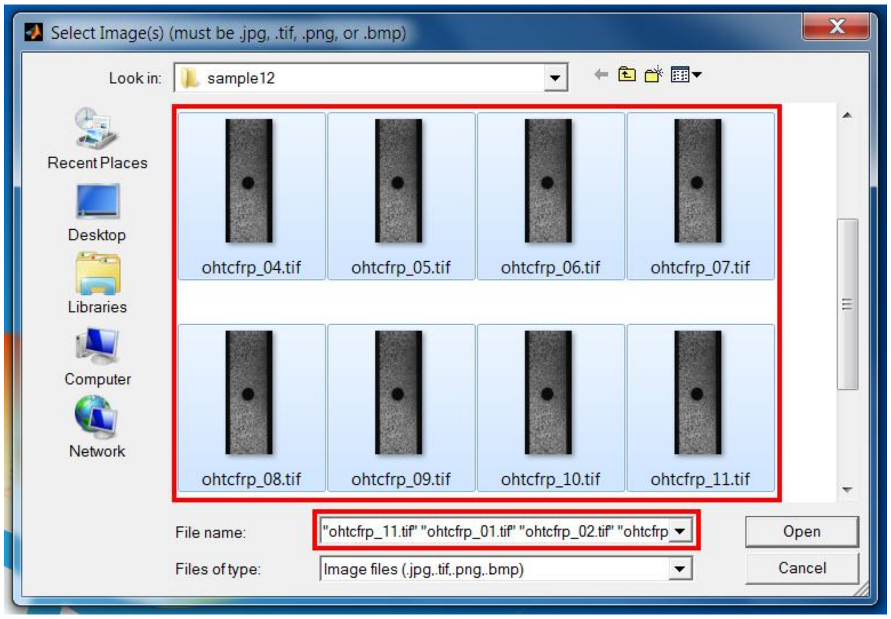
Where "name" is the name of the image set, "#" is a number associated with the image, and "ext" is the image extension, which must either be .jpg, .tif, .png, or .bmp (an example would be “sample_15.png”). Note that this naming convention is only required if more than one current image is loaded through the GUI.

Setting the current image(s) through the GUI also differs from the reference image, in that there are two different loading schemes, “Load All” and “Load Lazy,” which are shown by going to *File > Load Current Image(s)*.



“Load all” will store all the images into the workspace. If you are analyzing a large number of images, this will be problematic since it may result in running out of RAM space. However, when using lazy loading, only the name and path of the images are stored; the images are loaded based on their stored location “on demand.” This means that, when doing analysis in Ncorr using lazy loaded images, the user must make sure not to move the images into a different directory or alter them. Furthermore, since they are loaded on demand and then discarded afterward, this means some of the analyses may proceed a little slower. But, from my experience, the difference is not that noticeable. When in doubt, use lazy loading.

When loading through the GUI (for both “load all” and “lazy load”), the following window should appear:



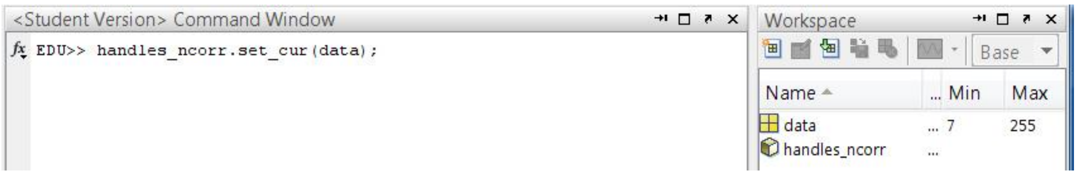
File name: "ohtcfrp_11.tif" "ohtcfrp_01.tif" "ohtcfrp_02.tif" "ohtcfrp

Files of type: Image files (.jpg,.tif,.png,.bmp)

NOTE: Take note of the naming convention above. All of the current images follow the "name_#.ext" format, where name = "ohtcfrp" and ext = "tif".

Simply drag and select all the current images you want to process. For this example, the images with a post-fix of "01" to "11" are selected).

Similar to the reference image, there is also an option to set the current image(s) directly through the Ncorr handle. To do this, you can type "handles_ncorr.set_cur(data)" through the MATLAB terminal, as shown below:

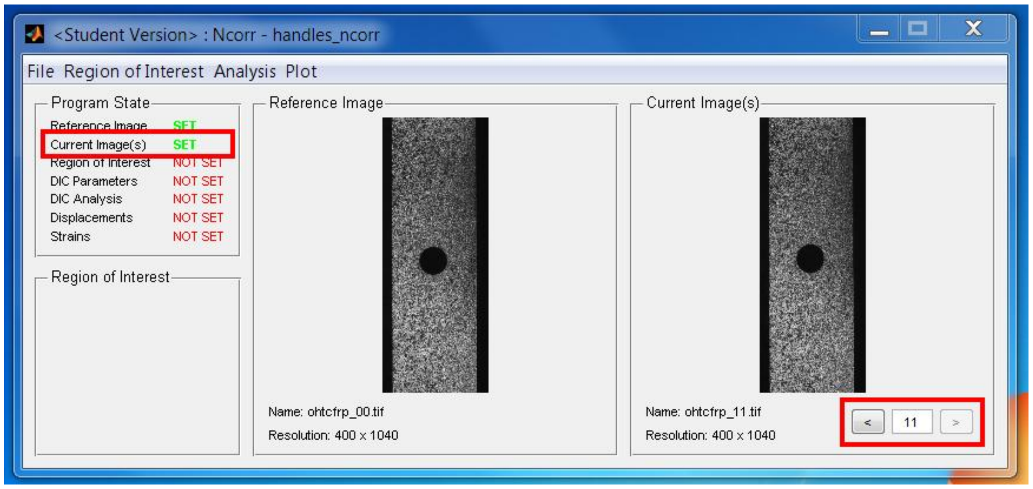


```
EDU>> handles_ncorr.set_cur(data);
```

Name	...	Min	Max
data	...	7	255
handles_ncorr	...		

In the figure above, only a single current image is loaded, so it exists as a grayscale array of type double, uint16, or uint8. To load more than one image, each current image gray scale array must be stored together in a *cell array*. Furthermore, the ordering specified in the cell array is the order that Ncorr uses to sort the images.

After the current images are uploaded, there are buttons on the bottom right of the Ncorr GUI which become enabled and allow you to scroll through the images as shown below:


<Student Version> : Ncorr - handles_ncorr
File Region of Interest Analysis Plot
Program State
Reference Image
Current Image(s)
Region of Interest

Reference Image	SET
Current Image(s)	SET
Region of Interest	NOT SET
DIC Parameters	NOT SET
DIC Analysis	NOT SET
Displacements	NOT SET
Strains	NOT SET

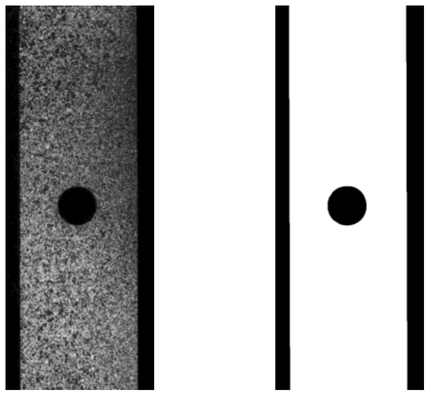
Name: ohtcfrp_00.tif
Resolution: 400 x 1040

Name: ohtcfrp_11.tif
Resolution: 400 x 1040

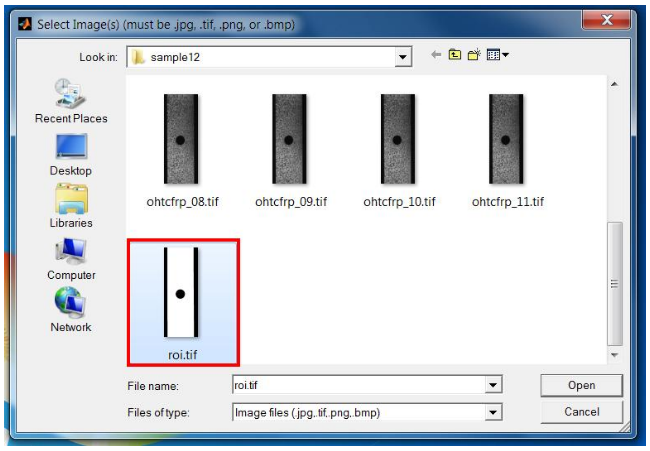
The current images should be ordered properly according to their number, and the last image in the sequence will appear initially by default. It's a good check to scroll through them to ensure the order is correct.

2.4 - Setting the Region of Interest

There are two ways to set the region of interest (ROI): through the GUI or through the MATLAB terminal. The most important thing to note here is that the ROI should be an array of the same size as the reference image or current image (this depends on which type of analysis is done, which is explained in greater detail in later sections. For this example, the ROI is set with respect to the *reference* image). Furthermore, white regions represent the ROI, and black regions are outside the ROI. It is also recommended to leave a black border near the edges of the image, although the ROI used in this example doesn't, in order to illustrate a couple points.


Sample ROI

Load ROI: The first way to set the ROI is to load it from an image by going to: *Region of Interest > Set Reference ROI* and then pressing the *Load ROI* button. This is the recommended way to set a ROI because you can use a program like Photoshop to accurately trace an outline of the sample in order to obtain the best ROI. The only problem with this method is that it can be tedious. Anyway, a GUI should appear as shown below:

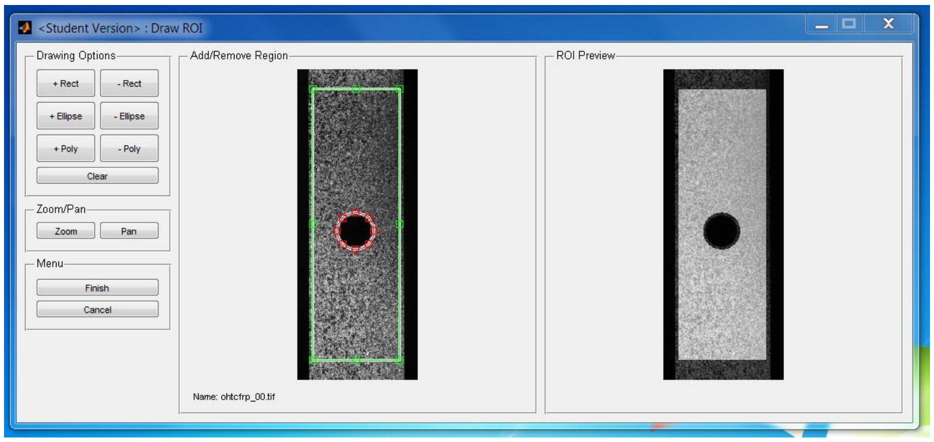


File name: roi.tif

Files of type: Image files (.jpg,.tif,.png,.bmp)

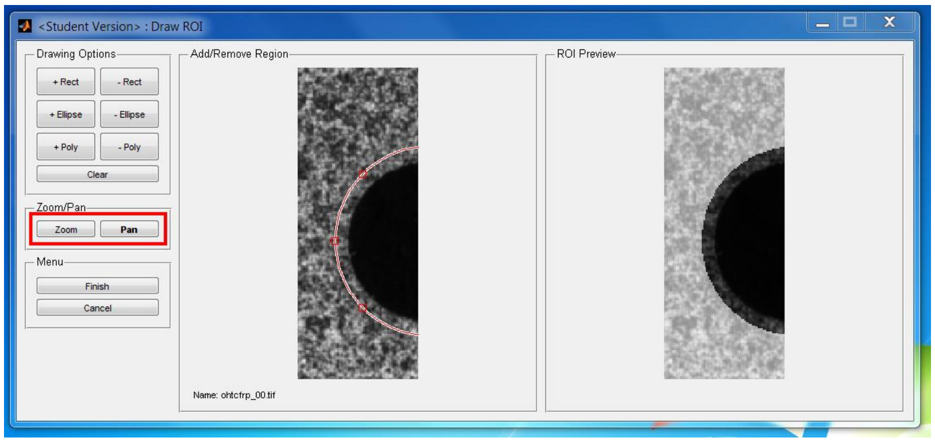
For the example used in this user guide, the ROI was set using this method.

Draw ROI: The second way to set the ROI is to draw the ROI directly in MATLAB by using *Region of Interest > Set Reference ROI* and then pressing the *Draw ROI* button. This is the preferred method for preliminary analysis because it can be done quickly. An example ROI is shown below and was constructed using "+ Poly" followed by "- Ellipse," where the "+" prefix indicates adding portions to the ROI; "-" subtracts regions:

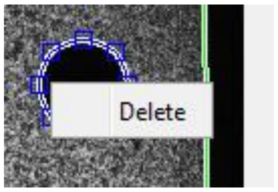


NOTE: The green shapes represent regions which add to the ROI, red shapes represent regions that subtract from the ROI, and blue shapes represent regions which are active (i.e clicked on)

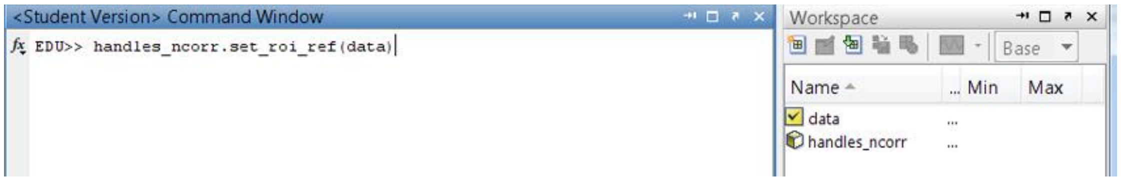
To obtain better precision when drawing ROIs, you can zoom and pan the view. Zooming can be enabled by clicking the zoom button once and disabled by clicking it again (or by clicking a different button, such as adding a region or pressing the pan button). An example is shown below:



It is also possible to delete specific shapes by right clicking them and selecting "Delete," as shown below:



MATLAB Terminal: Once again, similar to loading the reference and current image through the MATLAB terminal, typing "handles_ncorr.set_roi_ref(data)" will set the ROI. Note that data must be of type logical and be the same size as the reference image. An example is shown below:



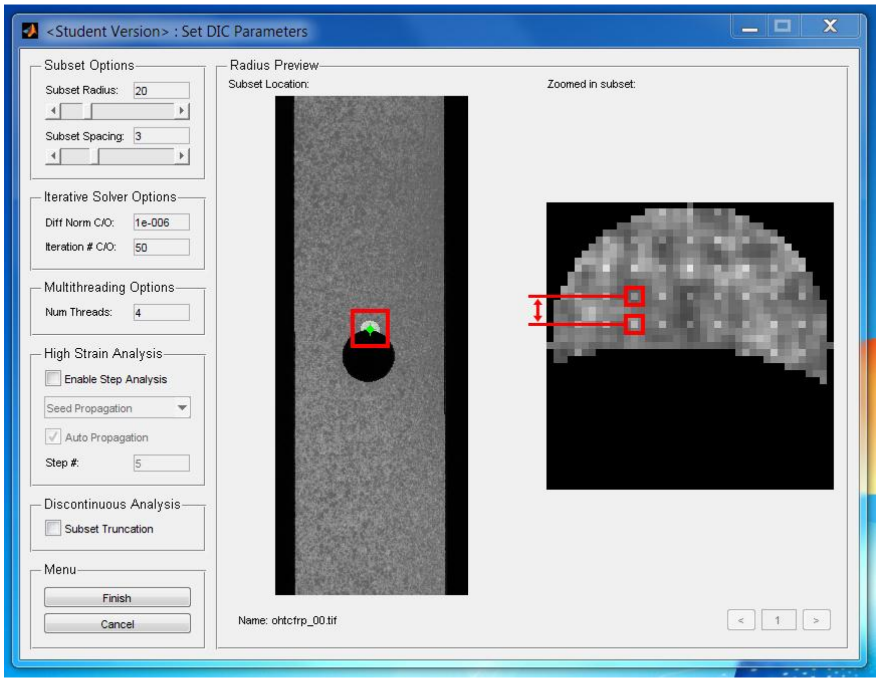
```
EDU>> handles_ncorr.set_roi_ref(data)
```

```
data            ...
handles_ncorr   ...
```

2.5 – Setting DIC Parameters

The DIC parameters can be set by going to *Analysis > Set DIC Parameters*. The main DIC algorithm used in Ncorr is based off of Bing Pan's RG-DIC framework. RG-DIC is highly robust and computationally efficient as well. More details can be found in Bing Pan's paper on RG-DIC as well as the "DIC Algorithms" section on the ncorr.com website.

A GUI should appear as shown below:



<Student Version> : Set DIC Parameters

Subset Options
Subset Radius: 20
Subset Spacing: 3

Iterative Solver Options
Diff Norm C/O: 1e-006
Iteration # C/O: 50

Multithreading Options
Num Threads: 4

High Strain Analysis
☐ Enable Step Analysis
Seed Propagation
✓ Auto Propagation
Step #: 5

Discontinuous Analysis
☐ Subset Truncation

Name: ohtcfrp_00.tif

NOTE: A subset spacing of 3 is shown above to better illustrate the spaced points on the zoomed in subset, but for the example analyzed in this section, a subset spacing of 1 was used to obtain higher resolution results.

There are several key components to this GUI. The first is obviously the menus on the left, but it is also important to note that the subset preview is interactive. A green *impoint* (highlighted by a red square) is placed in the axes labeled "Subset Location." This point is draggable and is the center point of the subset shown on the right. The subset on the right gives an idea of what the subset spacing (space between the two dots within the red squares) and subsets will appear like. It's important to note that these highlighted points are where the subset locations will be, and not part of the speckle pattern in the uploaded image.

Subset Options: These options are the main components of the DIC analysis. They dictate how large the subsets should be and the spacing between them. The spacing component is purely for reducing computational load. There are defaults for both options, but it's up to the user to select the most optimal settings. The most important option to get correct is the subset radius. There is an abundance of literature available for the selection of the subset size, as well as the effects of subset size on DIC analysis, but most of the conclusions from these studies are often based on heuristics and empirical observations. Overall, the main idea is to select **the smallest subset possible which does not result in noisy displacement data** (as larger subsets tend to have a smoothing effect). Some iteration may be required to get this option right.

Iterative Solver Options: The iterative solver used in the DIC analysis is the inverse compositional image alignment technique (A good paper on this topic is Baker et al and Bing Pan's paper on his adaptation for this technique for DIC). The exit criteria for this iterative solver are the norm of the difference vector as well as the number of iterations. The default options for the iterative solver are actually pretty strict, but they can be relaxed if faster analysis is desired by *reducing* the iteration number cutoff as well as *increasing* the norm of the difference vector cutoff.

Multithreading Options: Ncorr has the ability to use multithreading to speed up the computation process. The default number of threads is the number of cores specified by the user during installation (section 1), although I have left the option to alter this number again through this GUI.

High Strain Analysis: In the example used specifically in this section, high strain analysis is not required. However, the high strain analysis works by updating the reference image (as well as the ROI) and then “adding” displacement fields together. If you enable this parameter, there are two different ways the updating can proceed: In the case of the “seed propagation” option, the reference image will be updated based on the correlation coefficient and the number of iterations to convergence of the seeds. If these exceed certain heuristic thresholds, then the reference image will be updated. For the “leap frog” approach, the user can manually select how many images to use before updating the reference image. For both options, if the “automatic propagation” checkbox is enabled, then the seeds will be placed automatically for the updated reference image based on their previous displacements and the DIC analysis will proceed automatically until completion. If this is unchecked, then the user will have to replace the seeds manually every time the reference image is updated.

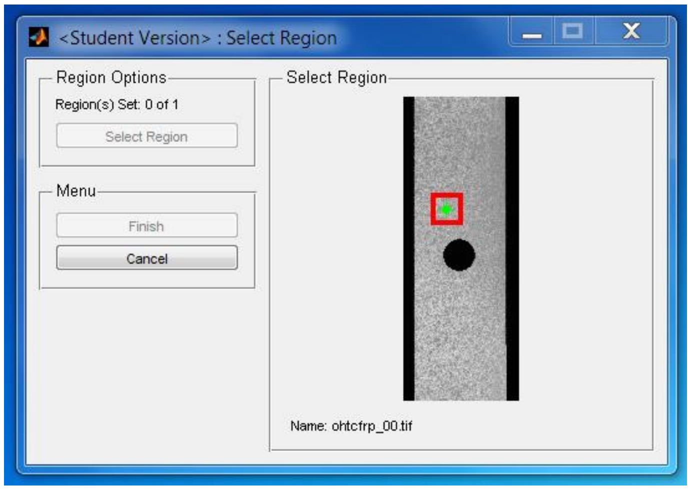
Discontinuous Analysis: For this example, discontinuous displacements are not anticipated, so this feature isn’t used. Regardless, subset truncation is a feature that prevents subsets from wrapping around a crack tip which can cause distortions near the cracktip. This feature is elaborated on in the discontinuous displacements section.

2.6 – DIC Analysis

The actual DIC analysis is performed by going to *Analysis > Perform DIC Analysis* in the main Ncorr GUI. The first step is to select a region (defined as a contiguous region) as shown below.

Select Region

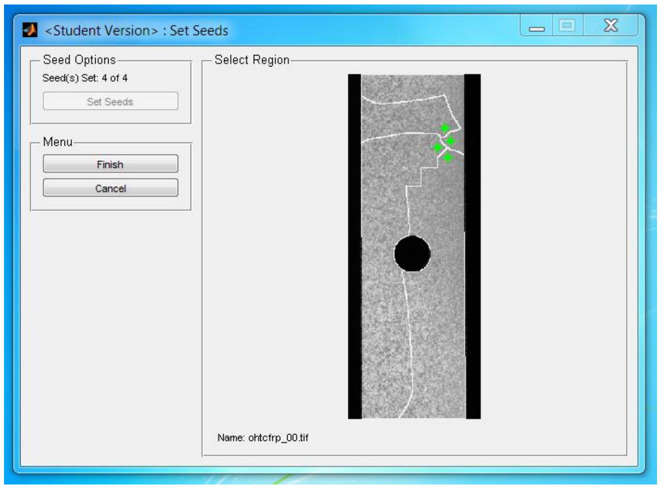
This step involves selecting a contiguous region to process. In this example, only one region will be processed (because only one is present), but it is possible to have more than one region when forming the ROI. Press the "Select Region" button and place the *impoint* anywhere within the region to select it for processing, as shown below:



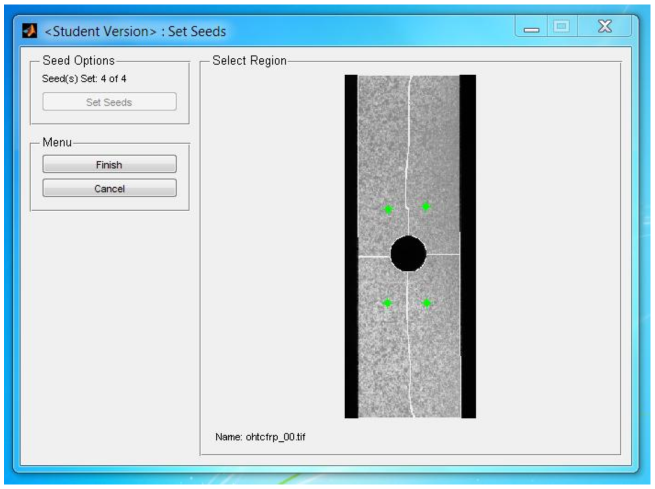
Seed Placement

The seed placement process serves three purposes: 1) it provides initial guesses for the RG-DIC analysis, 2) it partitions the ROI so that each partition can be calculated in parallel if multithreading is enabled, and 3) in the case of the high strain step analysis (with seed propagation selected), it updates the reference image based on certain heuristic thresholds for the iterations to convergence and correlation coefficients of the seeds. Thus, the placement of the seeds is very important. To satisfy the first requirement, seeds must be placed such that they do not go outside the field of view (FOV) during deformation. For the second requirement, seeds should be placed such that the ROI is partitioned evenly. For the third requirement, seeds should be placed in regions of high deformation such that the reference image updates appropriately. The first two points are addressed here, while the third is addressed in the high strain section.

An example of inappropriate vs appropriate seed placements for the example reference image is shown below (note that this is only applicable if multithreading is enabled – if it's disabled only one seed can be placed):



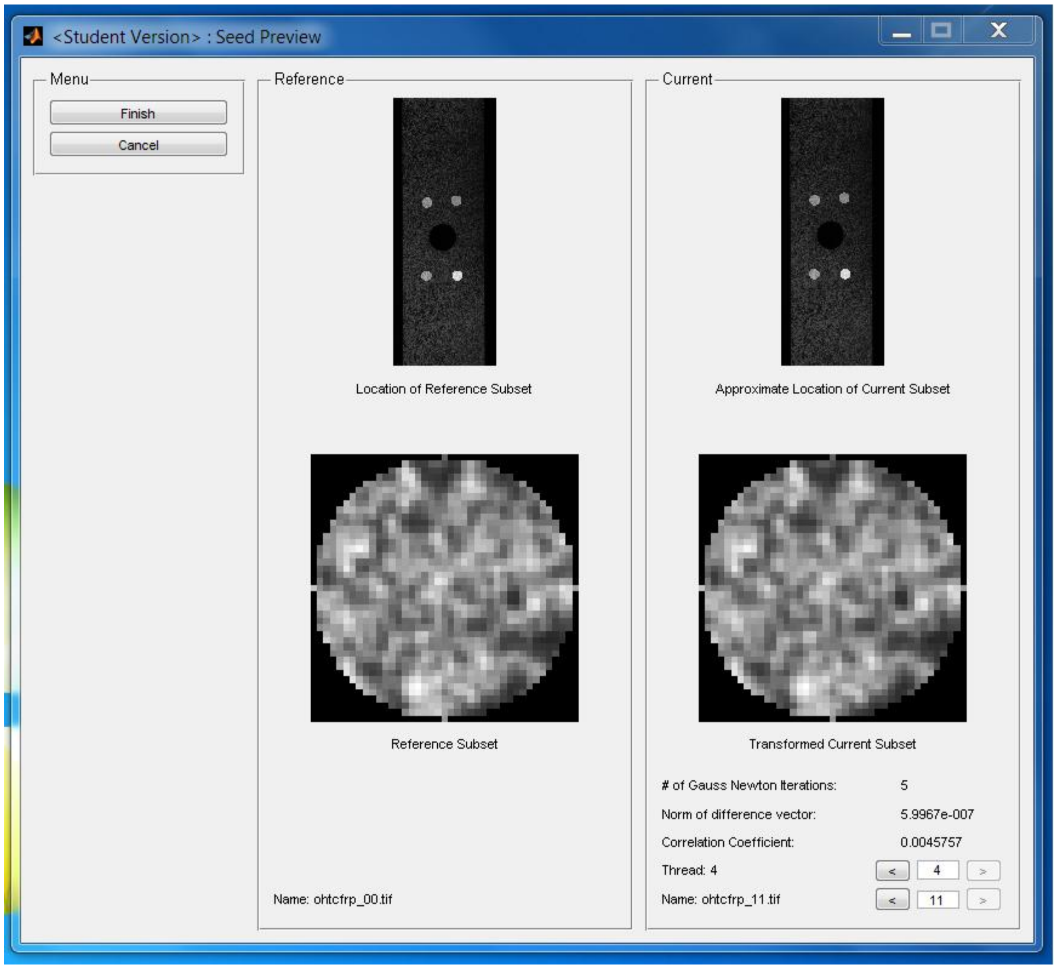
These seed positions are not preferable because the regions aren't evenly sized and they aren't nice "full" shapes.



These seed placements are more preferable because the regions are evenly sized and have nice "full" shapes.

Seed Preview

After the seed placement is done, a preview will appear. It's important that the user checks the seeds through this GUI because it's possible that the seeds can be processed incorrectly. The main reasons for seeds being incorrect can be either through a failure in convergence (this rarely happens as long as there are no large rotations or deformations present), or if the seeds travel outside the current image as the sample deforms. An example for proper seed preview is shown below:

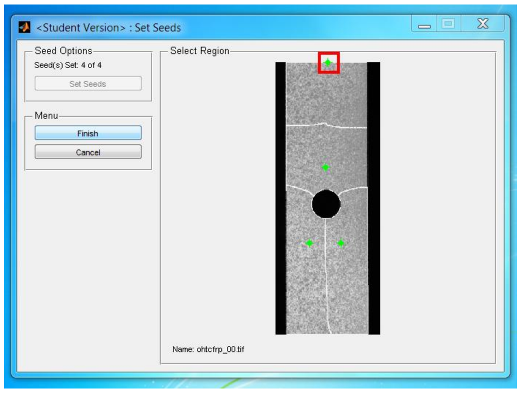

<Student Version> : Seed Preview
Menu
Finish
Cancel
Reference
Current
Location of Reference Subset
Approximate Location of Current Subset
Reference Subset
Transformed Current Subset

# of Gauss Newton Iterations: 5
Norm of difference vector: 5.9967e-007
Correlation Coefficient: 0.0045757
Thread: 4
Name: ohtcfrp_11.tif
Name: ohtcfrp_00.tif

Notice that all the seed locations in the reference image seem to match appropriately with the locations in the current image. Furthermore, the reference subset and transformed current subset look very similar. Lastly, the number of iterations to convergence was way below the cutoff, and the relatively low correlation coefficient both imply a good and correct seed placement.

Debugging

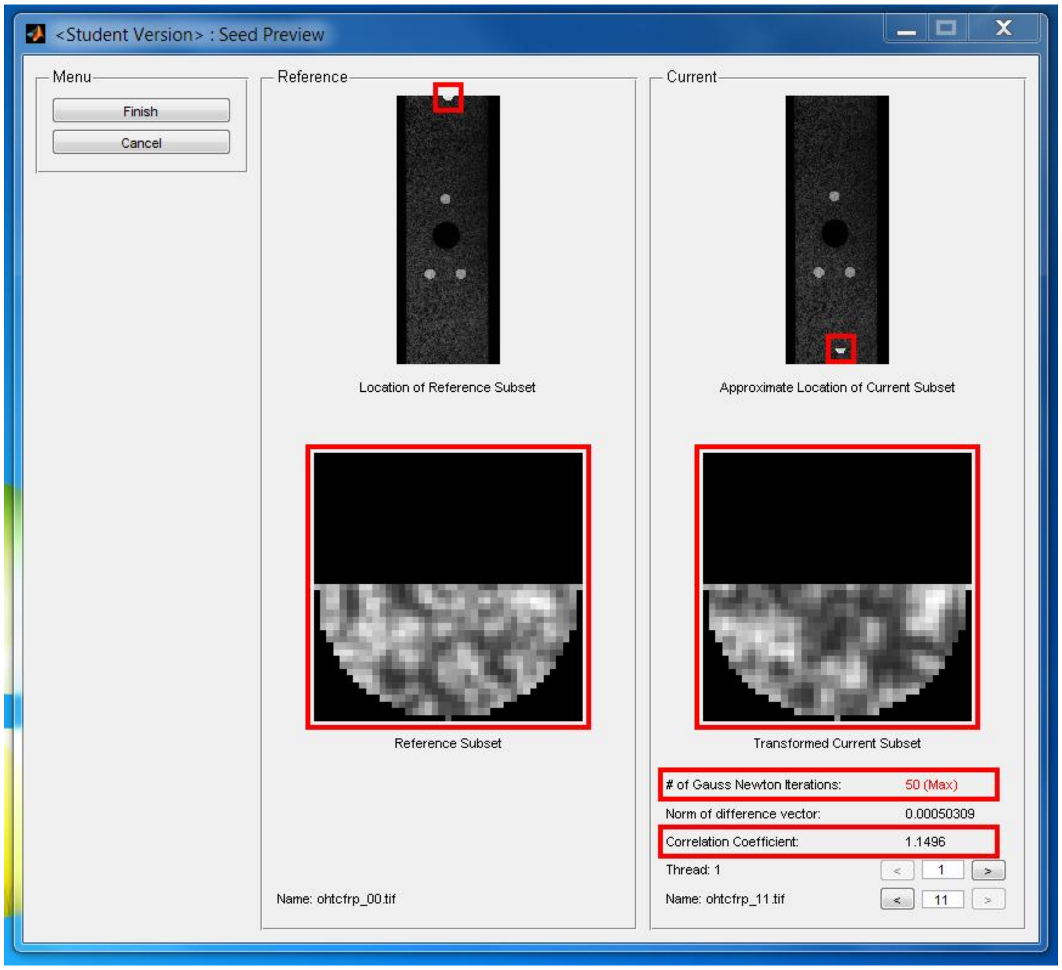
As mentioned before, it's possible for seeds to travel outside the image during deformation which results in improper seed calculations. Because this is actually a somewhat common occurrence (especially when the whole sample cannot be imaged within the field of view), an example is given below with tips on how to catch this mistake and correct it:



Seed(s) Set: 4 of 4

Name: ohtcfrp_00.tif

Suppose for instance that the seeds are placed as shown above. It turns out that the sample in the picture undergoes some translation upwards as it deforms. The seed highlighted in red actually goes "out of the picture" in some of the current images. In this case, an error message will generally display that notifies the user of a seed point with a high correlation coefficient value. This is an important indicator of a seed that has possibly been processed incorrectly. You can confirm this by viewing the seed preview below:


<Student Version> : Seed Preview
Menu
Finish
Cancel
Reference
Current
Location of Reference Subset
Approximate Location of Current Subset
Reference Subset
Transformed Current Subset

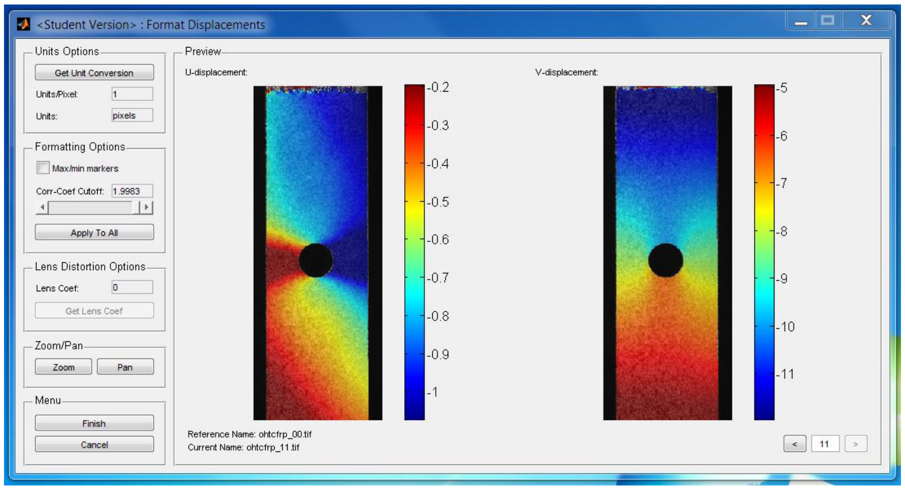
# of Gauss Newton Iterations: 50 (Max)
Norm of difference vector: 0.00050309
Correlation Coefficient: 1.1496
Thread: 1 — 1
Name: ohtcfrp_00.tif
Name: ohtcfrp_11.tif — 11

All of the highlighted things in the figure above indicate potential problems. In the location pictures, it becomes evident that the location of the top seed in the current image is completely off, whereas the other three seeds are correct. Visual inspection of the enlarged reference and current subsets also show that they look very different. Lastly, the maximum number of iterations (50 for this case) was used before convergence was reached; this is typically another good indicator of an improperly calculated seed. The solution here is to move the seed to a lower location or potentially redraw the ROI to exclude the top region.

Anyway, after the seeds have been properly placed and calculated, the RG-DIC analysis will run until completion. The next step in the process is to format the displacements.

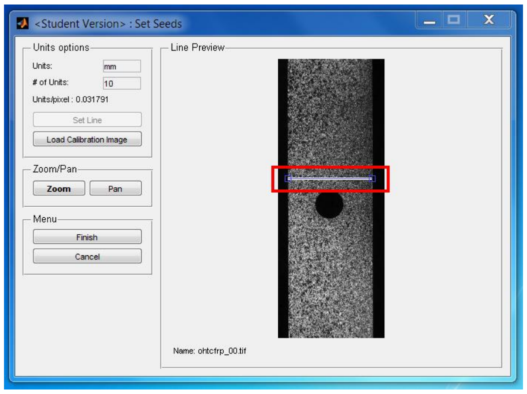
2.7 - Formatting Displacements

The displacement formatting begins by going to *Analysis > Format Displacements* in the main Ncorr GUI. The resulting GUI for formatting the displacements should look like this:



Units/Pixel: 1
Units: pixels
Corr-Coef Cutoff: 1.9983
Lens Coef: 0
Reference Name: ohtcfrp_00.tif
Current Name: ohtcfrp_11.tif
11

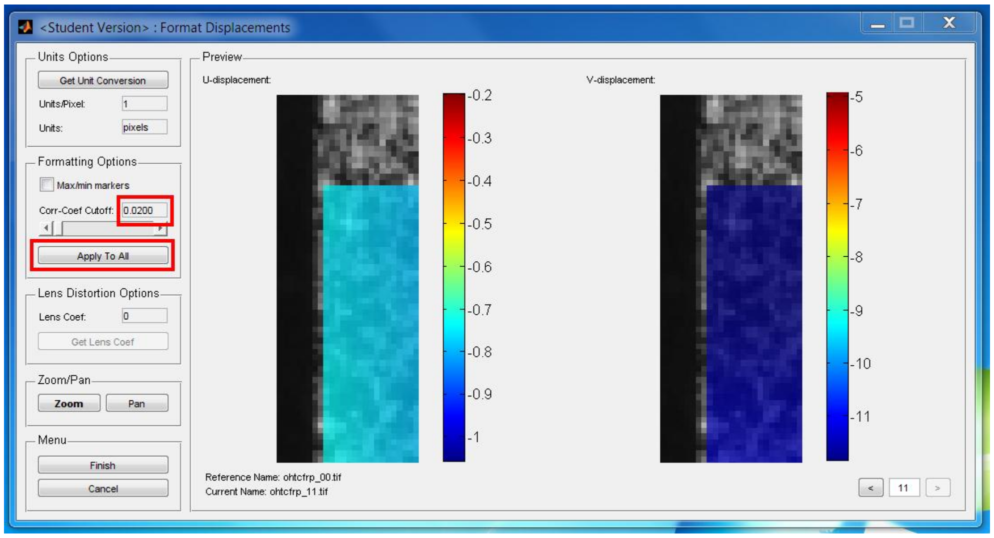
Units Options: These options are used to convert the displacements from pixels to real units. You can either input the value directly if you know it, or you can click the "Get Unit Conversion" button in order to load an image with a known dimension. Most tests are done with a regular camera on a sample, so the easiest way to take advantage of this feature is to, after the experiment in complete, replace the sample with a ruler so you can measure what the unit conversion is. An example of the GUI is shown below:



Units: mm
# of Units: 10
Units/pixel : 0.031791
Name: ohtcfrp_00.tif

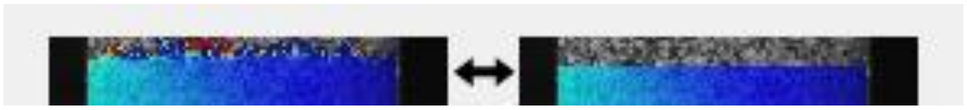
For this sample set, I'm actually not sure what the unit to pixel conversion is, so I've left the units as pixels. But, just for the sake of an example, assume that the undeformed sample is 10 mm wide. You can apply this knowledge by loading the reference image through "Load Calibration Image" and then setting a line across the sample with "Set Line." Lastly, the units (mm) and the number of units (10) can be set in their appropriate boxes. Assuming the sample is 10mm wide, this will apply the correct conversion to convert the displacements from pixels to mm (determined to be 0.031791 mm/pixel in this case). It's important to note that this assumes pixels are square, which is usually the case.

Formatting Options: These options help filter out “bad” data points. In the example done in this section, some of the points in the ROI travel outside the FOV. These points can be filtered out by reducing the correlation coefficient cutoff. In conjunction with this, the locations of the min and max displacement values can assist in this analysis, as “bad” data points generally have very high or very low values. A good cutoff for the example done in this guide is to set the correlation coefficient cutoff to 0.02 (which was applied to all images), as shown below:



Units/Pixel: 1
Units: pixels
Corr-Coef Cutoff: 0.0200
Lens Coef: 0
Reference Name: ohtcfrp_00.tif
Current Name: ohtcfrp_11.tif

Note that the preview above was zoomed in using the zoom/pan options. The filtering out of bad data points is shown below:



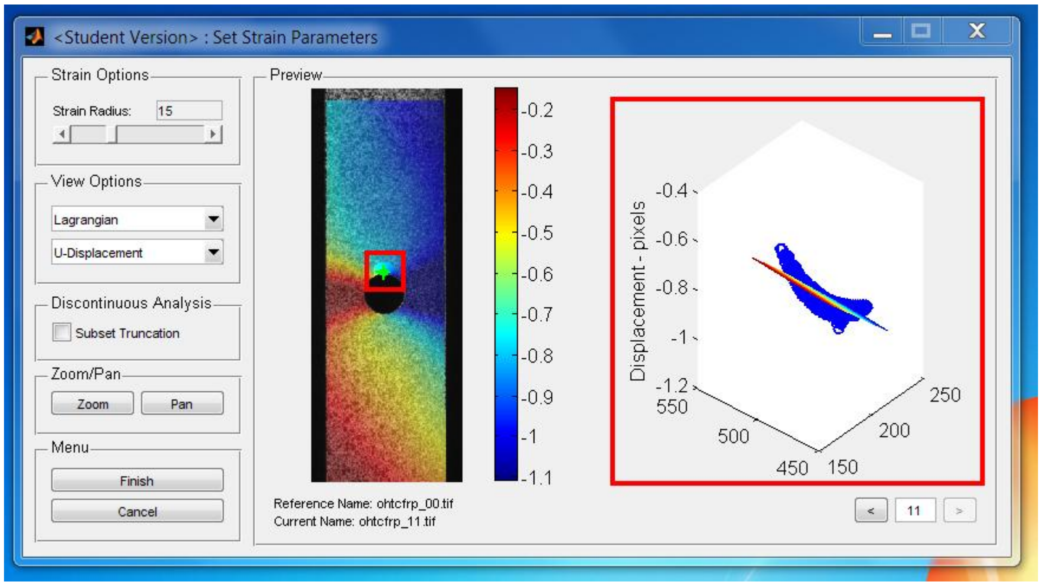
Lens Distortion Options: This is an option to correct for errors involving radial lens distortion. The distortion correction follows from eq.6 in “Systematic errors in two-dimensional digital image correlation due to lens distortion.” The correction is applied assuming the distortion center is at the center of the image; this might not always be true but is usually a reasonable assumption. If no calibration tests have been done or you’re unsure what the lens coefficient is, then just leave it as 0.

As of now, a lens coefficient can only be applied. Bing Pan provides a method for determining the lens coefficient, but I have yet to integrate this into Ncorr. The process basically requires DIC analysis to be run on images taken during rigid body translation tests and then performing a simple least squares analysis to solve for the lens distortion parameter. In future revisions of the program, I may implement this feature.

2.8 - Strain Analysis

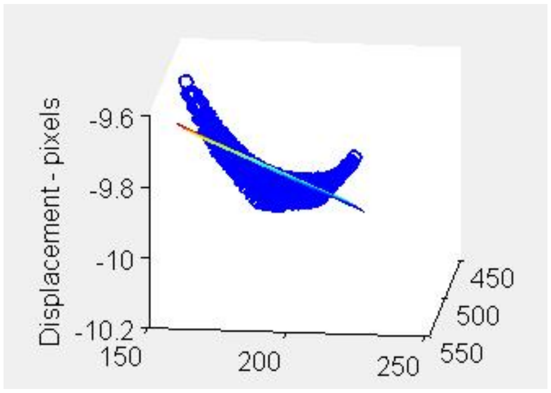
The strain analysis begins by going to *Analysis > Calculate Strains* in the main Ncorr GUI. The strains are calculated from the displacement data by using a least squares plane fit to a local group of data points (in this case a contiguous circle) which is based on Bing Pan's work on strain calculation. The displacement gradients are then found from the plane parameters; these gradients are used to calculate **Green-Lagrangian** and **Eulerian-Almansi** strains. A more detailed description is given in the "DIC Algorithms" section on the ncorr.com website.

Strain Options: The only parameter you can vary here is the strain radius. This is the radius of a circle which selects a group of points to fit a plane to. A preview is provided so the user can visualize the plane fitting. The selection of the ideal strain radius is similar to the selection of the ideal subset radius, in that **the smallest radius is desired which does not result in noisy strain data**. The default radius is set to 15, but it is up to the user to select the most optimal radius for their data.

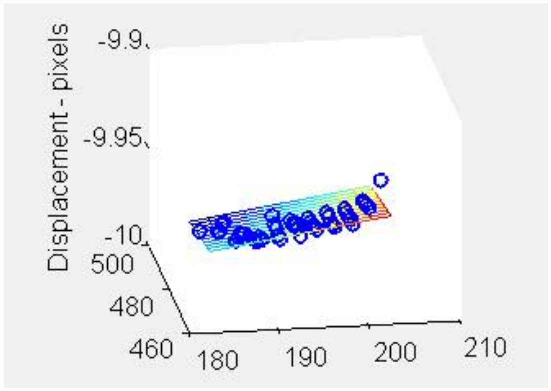


Reference Name: ohtcfrp_00.tif
Current Name: ohtcfrp_11.tif

The *impoint* highlighted in the red box is draggable, and updates the plane fit shown on the right side. The recommended way to use this analysis is to drag the point to areas of high deformation and view if the curve fit is still reasonable. An example of this is shown below:



Visually, it appears the strain radius is too large for this displacement field.



Reducing the strain radius to a value of 5 appears to give a much better fit for the plane. In this example, this value is used to generate the strain field.

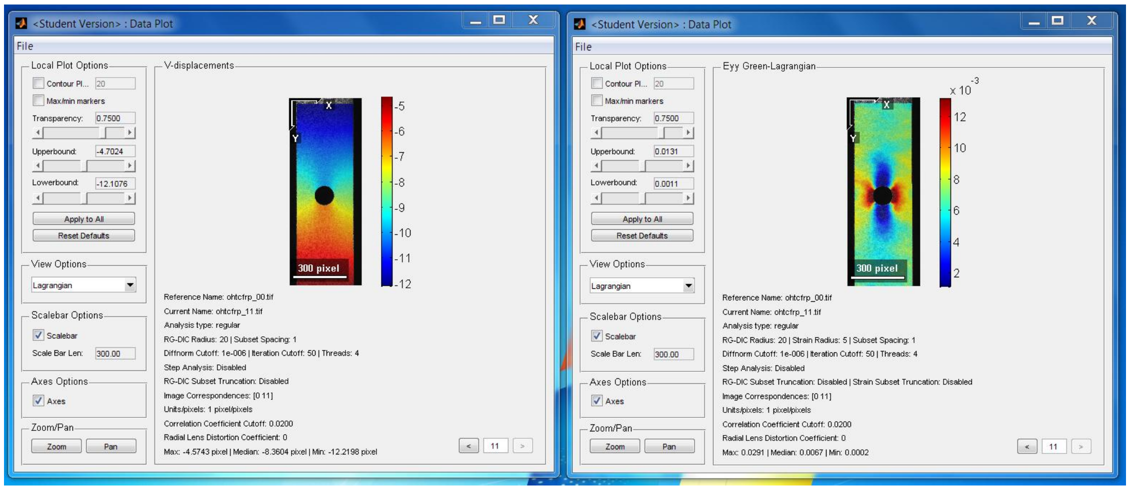
Lastly, the strain radius should be selected with regards to the last current image (which is shown first by default), as this image will most likely have the highest displacement gradients.

View Options: These dropdown menus allow you to switch from the U-displacements to the V-displacements and vice versa. They also allow you to switch from the Lagrangian to the Eulerian perspective. By default, both Green-Lagrangian and Eulerian-Almansi strains are calculated.

Discontinuous Analysis: Just like for DIC, there is also an option to utilize subset truncation for the strain calculations. This feature prevents subsets from wrapping around a crack tip which can cause distortions near the cracktip. This feature is elaborated on in the discontinuous displacements section.

2.9 - Plotting

At this point, it is assumed either the displacements have been formatted or the strains have been calculated. Plotting can be achieved by going to *Plot > View Displacement Plots* or *Plot > View Strain Plots* in the main Ncorr GUI. This is the last step of the analysis. An example of the plots used in the examples in this user guide is shown below:



Contour Pl... 20
Transparency: 0.7500
Upperbound: -4.7024
Lowerbound: -12.1076
Lagrangian
Scale Bar Len: 300.00

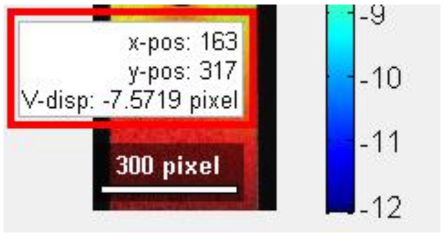
Reference Name: ohtcfrp_00.tif
Current Name: ohtcfrp_11.tif
Analysis type: regular
RG-DIC Radius: 20 | Subset Spacing: 1
Diffnorm Cutoff: 1e-006 | Iteration Cutoff: 50 | Threads: 4
Step Analysis: Disabled
RG-DIC Subset Truncation: Disabled
Image Correspondences: [0 11]
Units/pixels: 1 pixel/pixels
Correlation Coefficient Cutoff: 0.0200
Radial Lens Distortion Coefficient: 0
Max: -4.5743 pixel | Median: -8.3604 pixel | Min: -12.2198 pixel

Contour Pl... 20
Transparency: 0.7500
Upperbound: 0.0131
Lowerbound: 0.0011
Lagrangian
Scale Bar Len: 300.00

Reference Name: ohtcfrp_00.tif
Current Name: ohtcfrp_11.tif
Analysis type: regular
RG-DIC Radius: 20 | Strain Radius: 5 | Subset Spacing: 1
Diffnorm Cutoff: 1e-006 | Iteration Cutoff: 50 | Threads: 4
Step Analysis: Disabled
RG-DIC Subset Truncation: Disabled | Strain Subset Truncation: Disabled
Image Correspondences: [0 11]
Units/pixels: 1 pixel/pixels
Correlation Coefficient Cutoff: 0.0200
Radial Lens Distortion Coefficient: 0
Max: 0.0291 | Median: 0.0067 | Min: 0.0002

NOTE: From my experiences working with the plotting tool on different operating systems and versions of MATLAB, there appear to be inconsistencies between versions with regards to the stacking order of objects (i.e. sometimes the scalebar/axes appear behind the data if a contour plot is present), but I tried to optimize it for Windows on version R2012a. If the axes, scalebar, and contour plot are disabled, the plot appears to work fine on all versions of MATLAB I've tested Ncorr on.

There are a couple basic features available, the first one being that the plot windows are resizable. They are the only GUI windows in Ncorr that are resizable at the time of writing this manual. Furthermore, if you hover your cursor over the data, a data cursor appears that informs you of the displacement or strain data at that location, as shown below:



The “x-pos” and “y-pos” positions are the integer locations of the cursor *with respect to the raw data array*. This was done for convenience so the user can access the value in the array very easily. For example, in the figure above, the last V displacement plot is shown. You can access this point directly using:

```
handles_ncorr.data_dic.displacements(11).plot_v_ref_formatted(317,163)
```

The resulting output of the terminal will be -7.5719, which matches the V displacement shown. The index for the plot is the same that is shown between the "<" and ">" buttons on the bottom right of the plotting GUI. Note that these x and y positions are **not** the *real* positions of the x and y coordinates (hence why no units are provided). They are the position of the data cursor within the raw data array generated by Ncorr, which is a scaled multiple (the scaling depends on the subset spacing parameter) of the reference (or current - if eulerian analysis is done) image. For more information on the "data_dic" structure, please consult section 2.10.

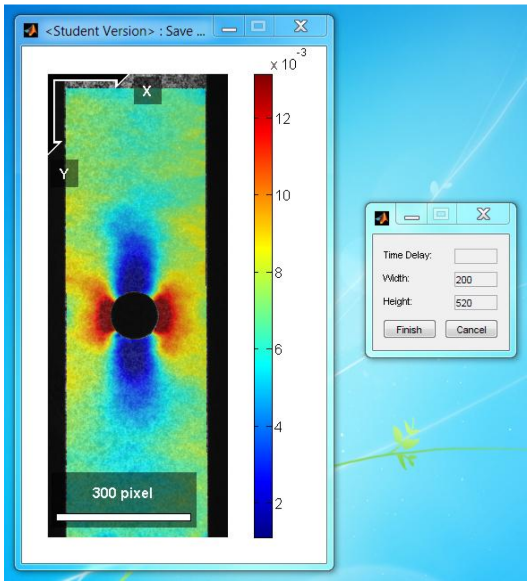
Local Plot Options: These options are pretty self-explanatory. The only thing to note here is that these options only affect the plot that is being adjusted (as opposed to other options which affect all open plots). Furthermore, the transparency option is disabled when the contour plot is enabled. This is because transparency does not work well with contour plots. In general, the contour plot does not work that well with strain plots because strain plots are generally noisier than displacement plots which results in jagged contours.

View Options: Once again, these options are pretty self-explanatory. These allow you to switch between Lagrangian and Eulerian perspectives. For strains, these will switch between Green-Lagrangian strains and Eulerian-Almansi finite strains tensors. This option will affect all open data plots.

Scalebar Options: This affects the visibility and length of the scalebar (displayed in the bottom-left of the plot). This option will affect all open data plots.

Axes Options: This option affects the visibility of the axes (displayed in the top-left of the plot). This option will affect all open data plots.

Saving the Figure: A figure can be saved by going to *File > Save Image > Save Image Without Info*, *File > Save Image > Save Image With Info, or File > Save Gif* **within the plot window**. This will allow you to save nice figures for publications, websites, or any other purpose. In this guide, I will focus on saving images, as the animated gif option is more useful for high strain calculation (which is discussed in the high strain section). After selecting the save image option, a menu with a preview should appear as so:

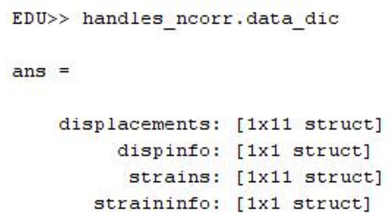


Time Delay:
Width: 200
Height: 520

This will allow you to resize the image if you desire. Please note that if a large spacing parameter is used, the image may look blurry if you upsize it because the image and data plot must be the same size when plotting in MATLAB. The best solution for this would be to rerun the analysis with a smaller subset spacing parameter. Also, I recommend saving the figure as a ".tif" because MATLAB appears to compress other image formats which may lead to a loss of quality.

2.10 - Directly Obtaining and Altering DIC Data

The DIC data can be accessed directly through "handles_ncorr." For example, by typing "handles_ncorr.data_dic" into the MATLAB terminal, you can access the data structure that contains the strain and displacement data as shown below:



```
EDU>> handles_ncorr.data_dic

ans =

    displacements: [1x11 struct]
         dispinfo: [1x1 struct]
          strains: [1x11 struct]
       straininfo: [1x1 struct]
```

Notice that both "dispinfo" and "straininfo" only have one element. This is because they contain parameters which are consistent over the entire analysis. On the other hand, both "displacements" and "strains" have eleven elements. This is because these fields contain information specific to each current image. For the example used in this section, we have uploaded eleven current images, which correspond to eleven elements seen above.

The size of the displacement/strain plots themselves depend on the image and the subset spacing utilized. For the Lagrangian perspective, the size of the displacement/strain plots will be:

```
size(img_ref(1:1+subsetspacing:end,1:1+subsetspacing:end))
```

where "img_ref" is the 2D reference image grayscale array and "subsetspacing" is the subset spacing parameter (where a value of 0 results in no reduction). For example, if a reference image of size 400x1040 and a subset spacing of 1 is used (as is done in the example in this manual), then the Lagrangian displacement/strain plots will be 200x520.

The displacements with respect to the reference configuration (Lagrangian) are stored in the plots with the "_ref" suffix. The "_cur" suffix is used for displacements with respect to the current configuration (Eulerian).

Each field of "handles_ncorr.data_dic" will be explained in a table format as shown below:

handles_ncorr.data_dic.displacements	
Field	Explanation
plot_u_dic	Plot containing U displacement data WRT the updated reference image (based on imgcorr) after DIC analysis. This plot is used for computational purposes.
plot_v_dic	Plot containing V displacement data WRT the updated reference image (based on imgcorr) after DIC analysis. This plot is used for computational purposes.
plot_corrcoef_dic	Plot containing correlation coefficient data WRT the updated reference image (based on imgcorr) after DIC analysis.
roi_dic	ROI corresponding to valid displacement data WRT the updated reference image (based on imgcorr) after DIC analysis.

plot_u_ref_formatted	Plot containing U displacement data WRT the reference configuration after it has been formatted. This plot is displayed in the plot figure.
plot_v_ref_formatted	Plot containing V displacement data WRT the reference configuration after it has been formatted. This plot is displayed in the plot figure.
roi_ref_formatted	ROI corresponding to formatted displacement data WRT the reference configuration
plot_u_cur_formatted	Plot containing U displacement data WRT the current configuration after it has been formatted. This plot is displayed in the plot figure.
plot_v_cur_formatted	Plot containing V displacement data WRT the reference configuration after it has been formatted. This plot is displayed in the plot figure.
roi_cur_formatted	ROI corresponding to formatted displacement data WRT the current configuration

handles_ncorr.data_dic.dispinfo

Field	Explanation
type	String representing whether “regular” or “backward” analysis was done; this value can either be 'regular' or 'backward'
radius	Integer representing the subset radius
spacing	Integer representing the subset spacing
cutoff_diffnorm	Number representing the cutoff for the norm of the difference vector in the inverse compositional method
cutoff_iteration	Integer representing the cutoff for the maximum number of iterations allowed for the inverse compositional method
total_threads	Integer representing the total number of threads specified during the RG-DIC parameter specification stage (can be different than the number of cores specified during installation)
stepanalysis	Structure containing data for the step analysis. The fields “enabled,” ”type,” “auto,” and “step” represent if the step analysis is enabled, whether seed propagation or leapfrog is used, whether seeds will automatically get placed at each step, and the step parameter for the leapfrog analysis, respectively.
subsettrunc	Logical parameter specifying if subset truncation was used for the RG-DIC analysis.
imgcorr	Structure containing the image correspondences for the step analysis. For each element, there is an “idx_ref” and “idx_cur.” The idx’s are zero based and assume the reference and current images are concatenated. idx_ref is the index for the updated reference image whereas idx_cur is the index of the last current image WRT this reference image.
pixtounits	Number representing the conversion between pixels to real units. In the form of units/pixels.
units	String containing the units
cutoff_corrcoef	Number representing the minimum correlation coefficient value allowable for displacement datapoints corresponding to a dataplot.
lenscoef	Number representing the radial lens distortion coefficient

handles_ncorr.data_dic.strains

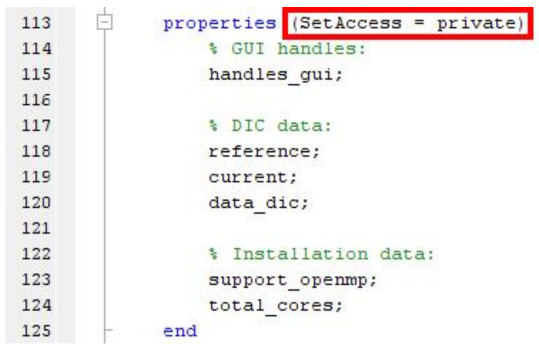
Field	Explanation
plot_exx_ref_formatted	Plot containing the Exx Green-Lagrangian strains (WRT the reference image). This plot is displayed in the plot figure.
plot_exy_ref_formatted	Plot containing the Exy Green-Lagrangian strains (WRT the reference image). This plot is displayed in the plot figure.
plot_eyy_ref_formatted	Plot containing the Eyy Green-Lagrangian strains (WRT the reference image). This plot is displayed in the plot figure.
roi_ref_formatted	ROI containing information for valid points after strain calculation (It's possible for some points to be removed during strain calculation for regions that do not have enough points to form a plane fit)

plot_exx_cur_formatted	Plot containing the Exx Eulerian-Almansi strains (WRT the current image). This plot is displayed in the plot figure.
plot_exy_cur_formatted	Plot containing the Exy Eulerian-Almansi strains (WRT the current image). This plot is displayed in the plot figure.
plot_eyy_cur_formatted	Plot containing the Eyy Eulerian-Almansi strains (WRT the current image). This plot is displayed in the plot figure.
roi_cur_formatted	ROI containing information for valid points after strain calculation (It's possible for some points to be removed during strain calculation for regions that do not have enough points to form a plane fit)

handles_ncorr.data_dic.straininfo

Field	Explanation
radius	Integer representing the strain radius
subsettrunc	Logical parameter specifying if subset truncation was used for the strain analysis.

Lastly, any information that the user wants to alter should be copied first to another variable in the main workspace. By default, the properties of Ncorr are set to "(SetAccess = private)" to prevent inadvertent alteration of data. If the user wants to alter the data directly, then the user can go into the **ncorr.m** source code and change "(SetAccess = private)" to "(Access = public)" to alter these properties directly as shown below:



```
properties (SetAccess = private)
    % GUI handles:
    handles_gui;

    % DIC data:
    reference;
    current;
    data_dic;

    % Installation data:
    support_openmp;
    total_cores;
end
```

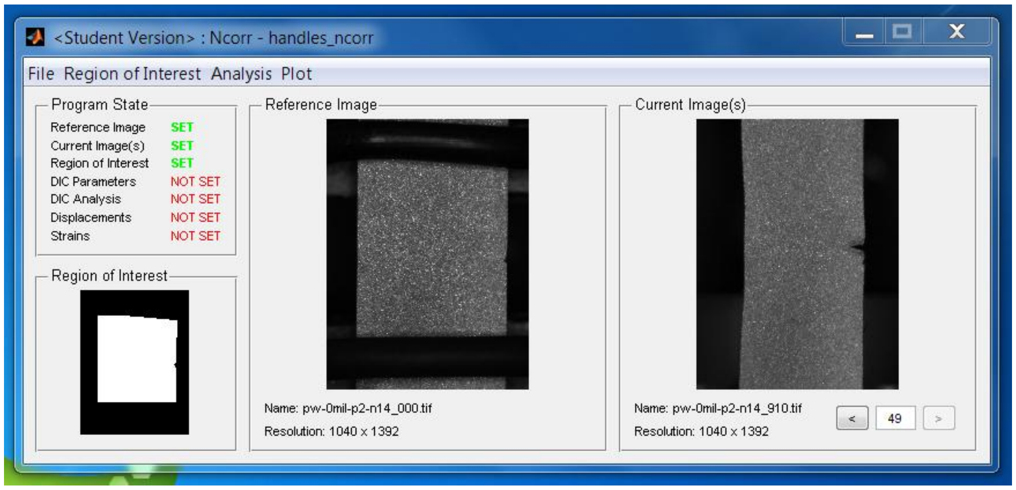
It may be possible to crash Ncorr by setting inconsistent data and thus caution is advised if going this route.

3.1 – High Strain DIC Analysis

For this section, it is assumed the program has already been installed and the user has already read the General DIC section. The images used for this section (the “weld” sample from SEM’s DIC challenge) are available off the Ncorr website if the user would like to follow along:

http://ncorr.com/download/sample13.zip

Start by setting the reference image to the first image (post fix of “000”) of the set. Next, load the current images by using “Lazy Load” (or potentially “Load All” if your computer has a lot of RAM) and selecting all the images from postfix of “020” until the postfix of “910” (note that at “915” the sample has failed- so omit this picture) as the current images. Lastly, set the *reference ROI* to the ROI provided in the image set (named “roi.tif”). If successful, the GUI should appear as below:


<Student Version> : Ncorr - handles_ncorr
File Region of Interest Analysis Plot
Program State
Reference Image
Current Image(s)
Region of Interest

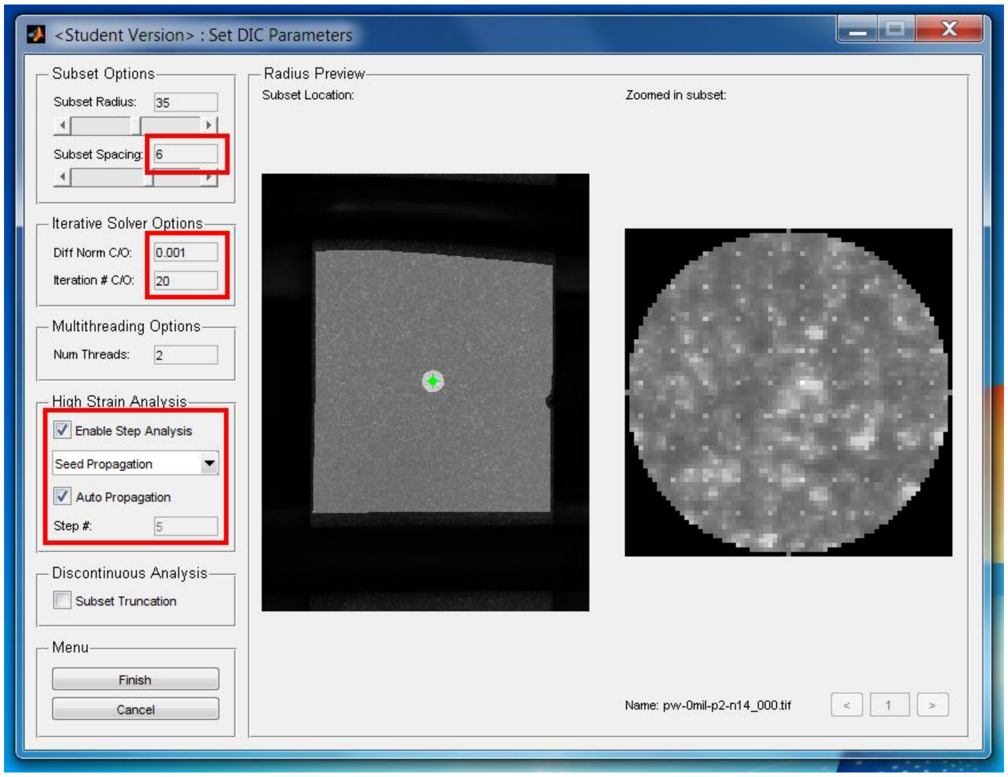
Reference Image SET
Current Image(s) SET
Region of Interest SET
DIC Parameters NOT SET
DIC Analysis NOT SET
Displacements NOT SET
Strains NOT SET

Name: pw-0mil-p2-n14_000.tif
Resolution: 1040 x 1392

Name: pw-0mil-p2-n14_910.tif
Resolution: 1040 x 1392

Now, if the user simply proceeds as is done in the general DIC section, he/she will find that the seeds for the latter images will always be incorrect for the latter images. This is because this sample undergoes very high deformation. The underlying problem here is that the pattern between the reference image and the last current image changes so much that the matching algorithm no longer works. This type of sample requires the reference image to be updated. To account for this, the high strain step analysis needs to be enabled when setting the DIC parameters; this is discussed in the next paragraph.

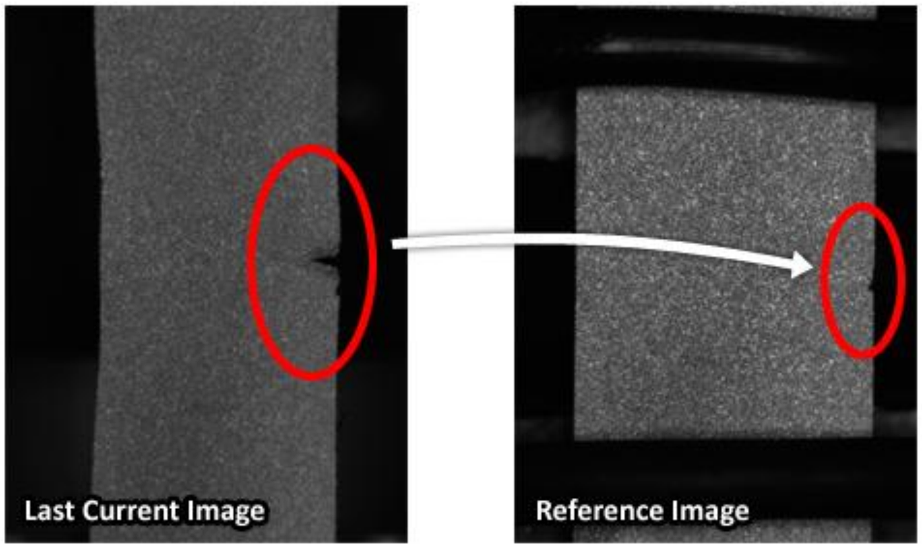
For high strain DIC, it’s recommended to set the type of step analysis to “seed propagation” with the “automatic propagation” checkbox enabled, as shown below:



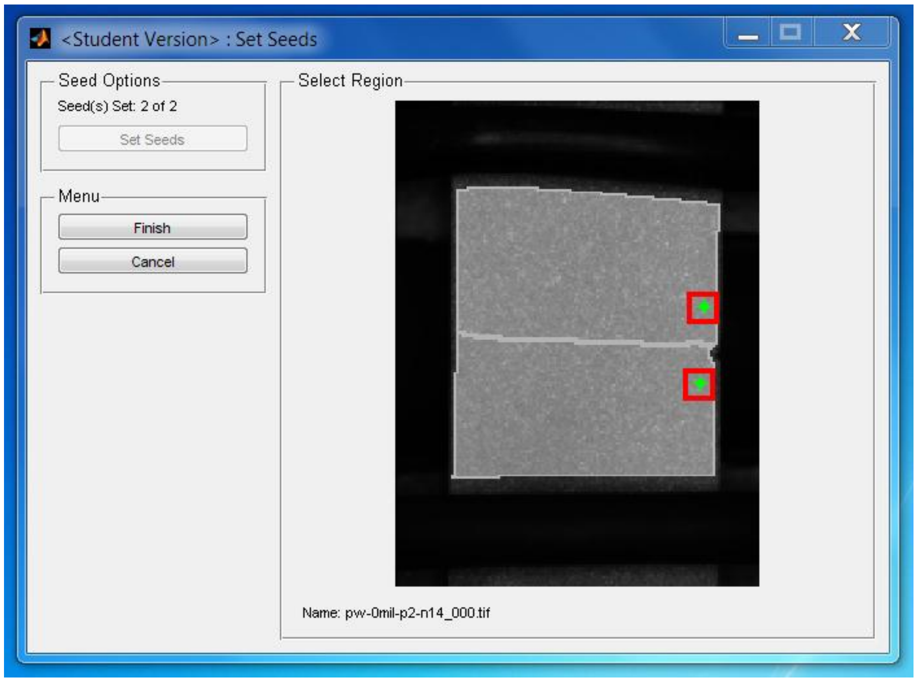
There are a couple things to note here. First of all, there are a large number of images involved in this analysis, so performance issues will be a concern. To increase the speed of the analysis, I've elected to increase the subset spacing to 6. Furthermore, the norm of the difference vector has been increased to 10^{-3} and the iteration cutoff has been reduced to 20. This will reduce the number of material points to analyze and also make the conditions on the iterative solver more relaxed. The effects of doing these three things will be a lowered resolution in the acquired displacement field, and a slightly reduced accuracy as well. If time permits, then the spacing, norm of difference vector, and iteration cutoff can be left to their default values.

In addition, the high strain analysis has been set with the seed propagation and automatic propagation parameters enabled. This allows the program to automatically update the reference image, as well as to automatically place seeds so the analysis can proceed in an automated fashion until completion. If "leapfrog" is selected, then the reference image will update according to whatever step number is provided. Lastly, if auto propagation is disabled, then the user will have to manually place the seeds every time the reference image is updated. This can be quite tedious, but also ensures the seeds are correct, so this option should only be disabled if you want to ensure completely that the seed placements are correct.

The next crucial step is the seed placements. Recall from the general DIC analysis that there are two main considerations for seed placement for the regular DIC analysis. First of all, they need to be placed on a material point that does not exit the FOV as the sample deforms. Next, they need to be placed such that the regions are more or less evenly portioned so they can be calculated nicely in parallel. For high strain analysis with seed propagation, there's another consideration: since the reference image is updated based on the correlation coefficient and number of iterations to convergence of the seed points, they need to be placed in a region of high deformation, so that the reference image updates correctly. If you examine the last current image in the example set, you'll see that the region of highest deformation is near the region circled below:



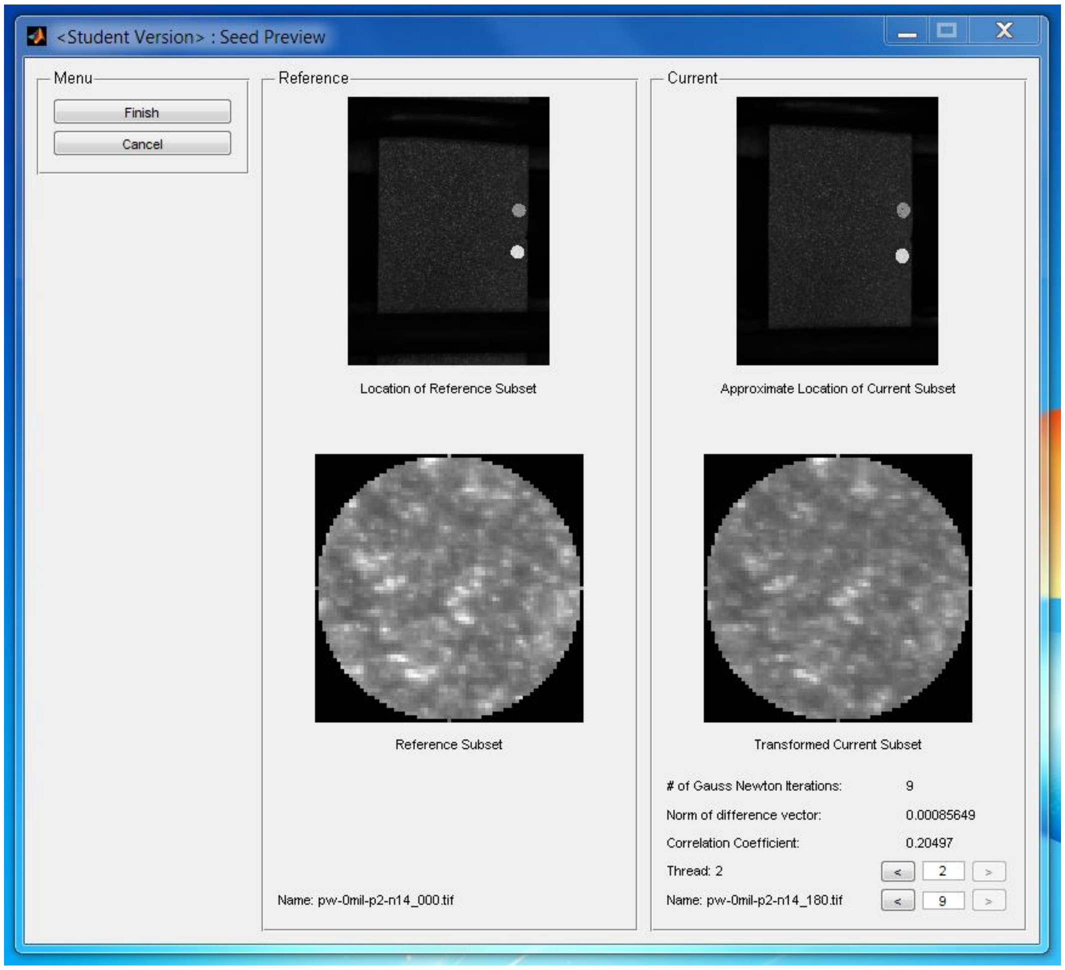
Basically, from inspection of the last current image (shown on the left), you can assume the highest deformation occurs approximately in the region encircled by the red ellipse (where the crack initiates). It would be ideal to place the seed points on the reference image corresponding to this region so that the reference image updates properly. With this in mind, the seeds (assuming multithreading is enabled with two threads) are placed as so:



Seed(s) Set: 2 of 2

Name: pw-0mil-p2-n14_000.tif

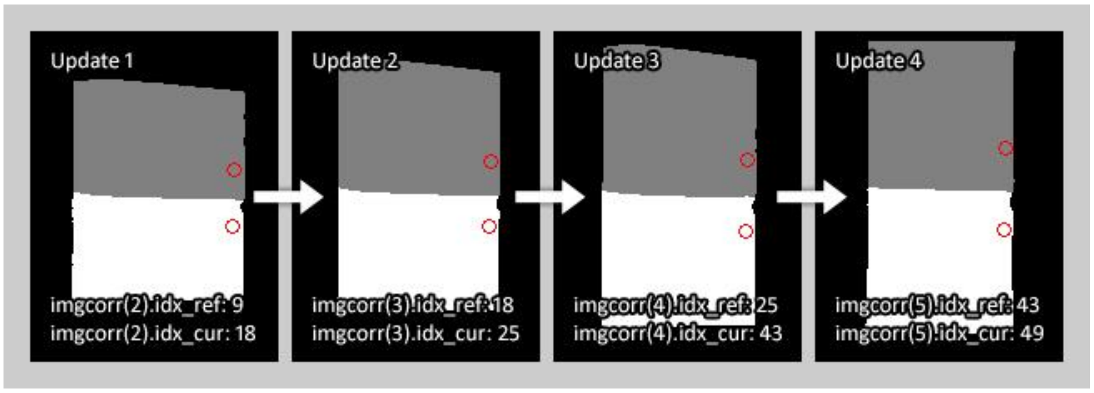
These seed locations (shown in the red boxes) satisfy the three requirements explained previously. Clicking finish yields the following seed preview screen:


<Student Version> : Seed Preview
Menu
Finish
Cancel
Reference
Current
Location of Reference Subset
Approximate Location of Current Subset
Reference Subset
Transformed Current Subset

# of Gauss Newton Iterations: 9
Norm of difference vector: 0.00085649
Correlation Coefficient: 0.20497
Thread: 2 — 2
Name: pw-0mil-p2-n14_000.tif
Name: pw-0mil-p2-n14_180.tif — 9

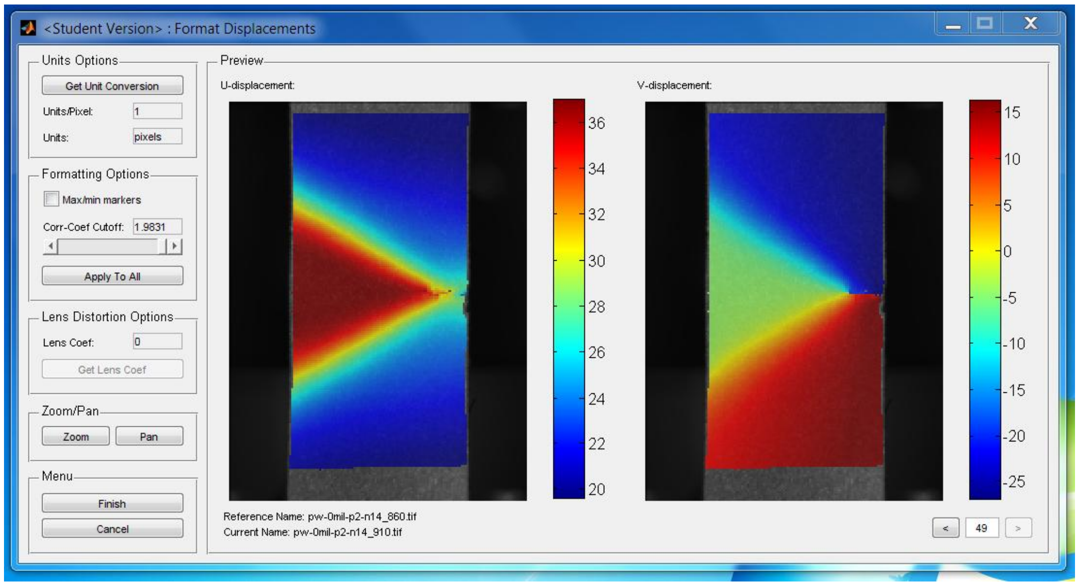
The most important thing to note here is that the last image seeded is not the last image of the series. The image with postfix “180” was determined to be the last properly seeded image of the series (Note that this can vary; it’s possible the image can be a couple before or after this one depending on the seed placement). This image will serve as the updated reference image until it is updated again. Click finish to proceed with the analysis. It should continue until completion.

Before continuing, I’d like to explain what exactly is going on “under the hood.” Basically, every time the reference image gets updated, the seeds are automatically “propagated” and placed as the analysis proceeds. Additionally, the ROI is also updated based on the displacements on the boundary of the ROI. For this example, the updated thread diagrams with seed placements are shown below:



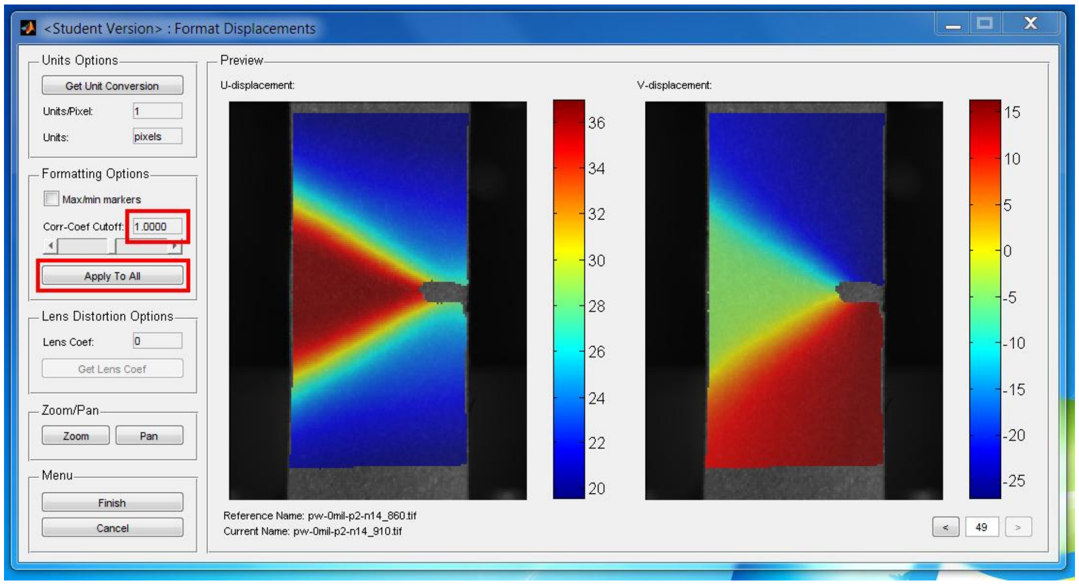
For this particular example, there were four updates. The ROIs and seed positions (shown as red circles) are automatically updated and then subsequently processed. The corresponding “imgcorr” structures are also superimposed as an example for how this structure works.

From here, the only problem is that the displacement fields are now going to be with respect to the updated reference images. The next step is to somehow “add” them in order to bring them all back to the Lagrangian perspective (i.e with respect to the original reference image). This step is taken care of when the displacements are formatted. The formatting displacements GUI should look similar to the one shown below:



Units/Pixel: 1
Units: pixels
Max/min markers
Corr-Coef Cutoff: 1.9831
Lens Coef: 0
Reference Name: pw-0mil-p2-n14_860.tif
Current Name: pw-0mil-p2-n14_910.tif

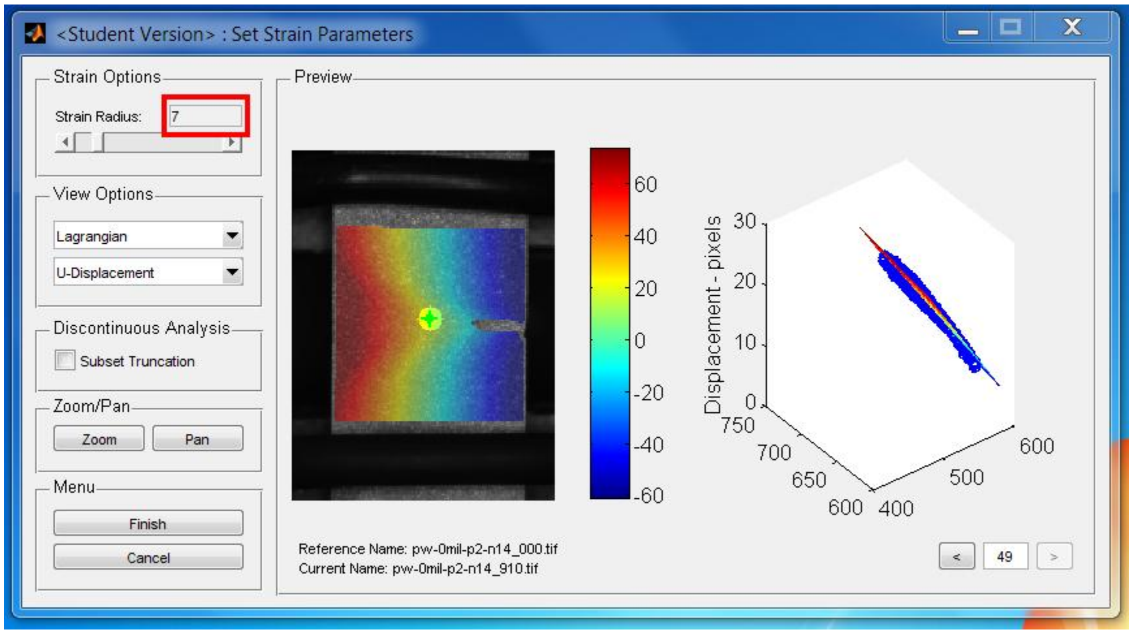
Note that a crack develops in the sample on the right side near the middle of the sample. This means points near this crack are poorly analyzed, so they should be removed. Setting a correlation coefficient cutoff of ~1.0 yields good results:



Units/Pixel: 1
Units: pixels
Corr-Coef Cutoff: 1.0000
Lens Coef: 0
Reference Name: pw-0mil-p2-n14_860.tif
Current Name: pw-0mil-p2-n14_910.tif

One last thing to point out is that the reference name shown above (near the bottom left) refers to the updated reference image.

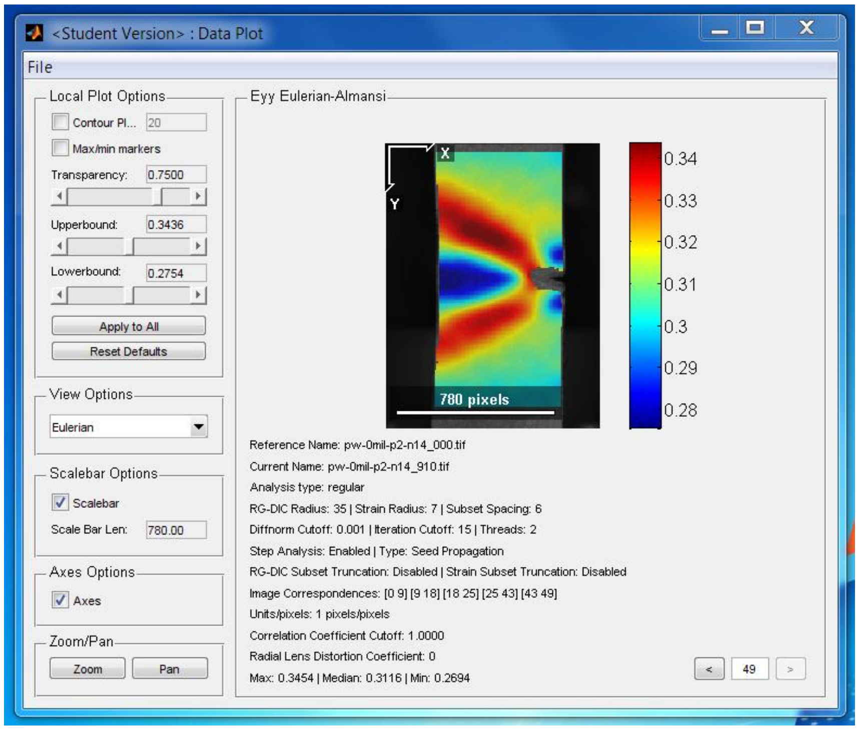
The last step of the analysis is to calculate the strains. A strain radius of 7 appears to give good results as shown below:



Strain Radius: 7
Lagrangian
U-Displacement
Reference Name: pw-0mil-p2-n14_000.tif
Current Name: pw-0mil-p2-n14_910.tif

In terms of plotting, the only distinction to really make between high strain and regular DIC analysis is that the user might want to consider saving an animated gif instead of regular pictures since so many data plots are available. In this particular case, the Eulerian perspective makes for especially good animations. Furthermore, it’s usually appropriate to set the same upperbound and lowerbounds for strain in all the plots.

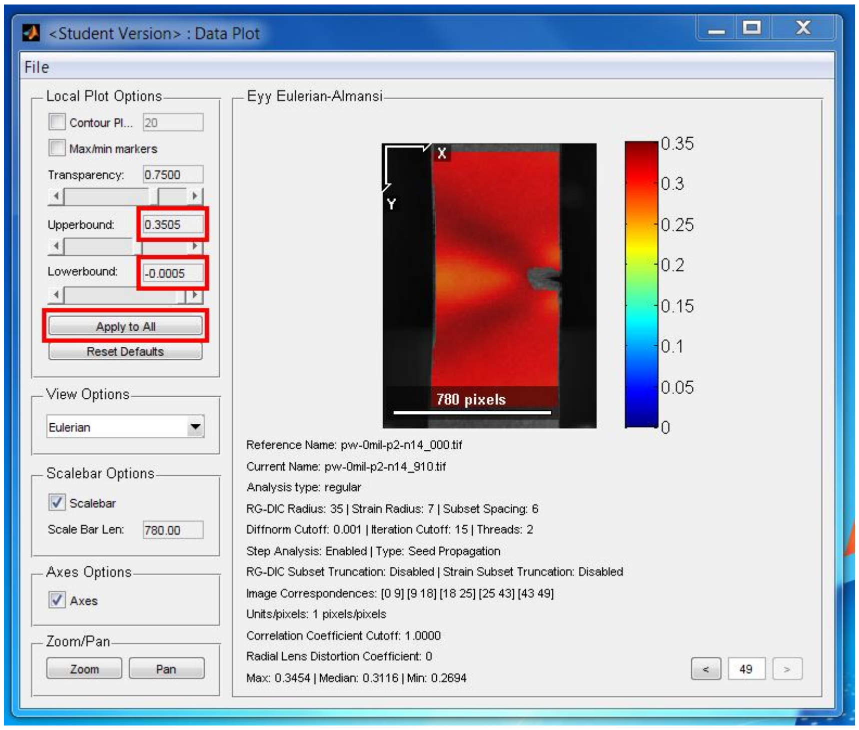
First, a plot for Eyy is shown below:



Contour Pl... 20
Transparency: 0.7500
Upperbound: 0.3436
Lowerbound: 0.2754
View Options: Eulerian
Scalebar: ✓
Scale Bar Len: 780.00
Axes: ✓

Reference Name: pw-0mil-p2-n14_000.tif
Current Name: pw-0mil-p2-n14_910.tif
Analysis type: regular
RG-DIC Radius: 35 | Strain Radius: 7 | Subset Spacing: 6
Diffnorm Cutoff: 0.001 | Iteration Cutoff: 15 | Threads: 2
Step Analysis: Enabled | Type: Seed Propagation
RG-DIC Subset Truncation: Disabled | Strain Subset Truncation: Disabled
Image Correspondences: [0 9] [9 18] [18 25] [25 43] [43 49]
Units/pixels: 1 pixels/pixels
Correlation Coefficient Cutoff: 1.0000
Radial Lens Distortion Coefficient: 0
Max: 0.3454 | Median: 0.3116 | Min: 0.2694

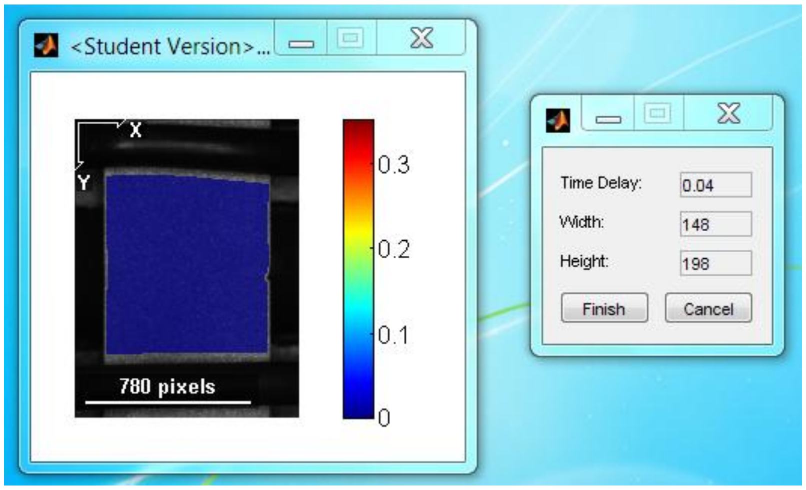
In order to set all the bounds for the strains the same, it’s necessary to set them such that the bounds accommodate all plots. For this example, the lowerbound must be set near zero as shown below:



Contour Pl... 20
Transparency: 0.7500
Upperbound: 0.3505
Lowerbound: -0.0005
View Options: Eulerian
Scalebar: ✓
Scale Bar Len: 780.00
Axes: ✓

Reference Name: pw-0mil-p2-n14_000.tif
Current Name: pw-0mil-p2-n14_910.tif
Analysis type: regular
RG-DIC Radius: 35 | Strain Radius: 7 | Subset Spacing: 6
Diffnorm Cutoff: 0.001 | Iteration Cutoff: 15 | Threads: 2
Step Analysis: Enabled | Type: Seed Propagation
RG-DIC Subset Truncation: Disabled | Strain Subset Truncation: Disabled
Image Correspondences: [0 9] [9 18] [18 25] [25 43] [43 49]
Units/pixels: 1 pixels/pixels
Correlation Coefficient Cutoff: 1.0000
Radial Lens Distortion Coefficient: 0
Max: 0.3454 | Median: 0.3116 | Min: 0.2694

If this lowerbound is set any higher, it might result in an earlier plot becoming completely saturated. If you do this, an error message will appear which informs you what the bounds must be. Furthermore, I tend to add or subtract 0.0005 from the bounds to guarantee the tick marks appear on the top and bottom of the colorbar as well, as shown above. Anyway, after hitting “applying to all”, select *File > Save GIF* which will bring up the following GUI:



Time Delay: 0.04
Width: 148
Height: 198

Notice that, in addition to the size, you can also modify the time delay between animation frames. From my experience, a time delay of 0.04 (25 fps) seems to work well for this data set, but the user can always opt to increase it in order to slow the animation down and vice-versa.

4.1 – Discontinuous Displacements DIC Analysis

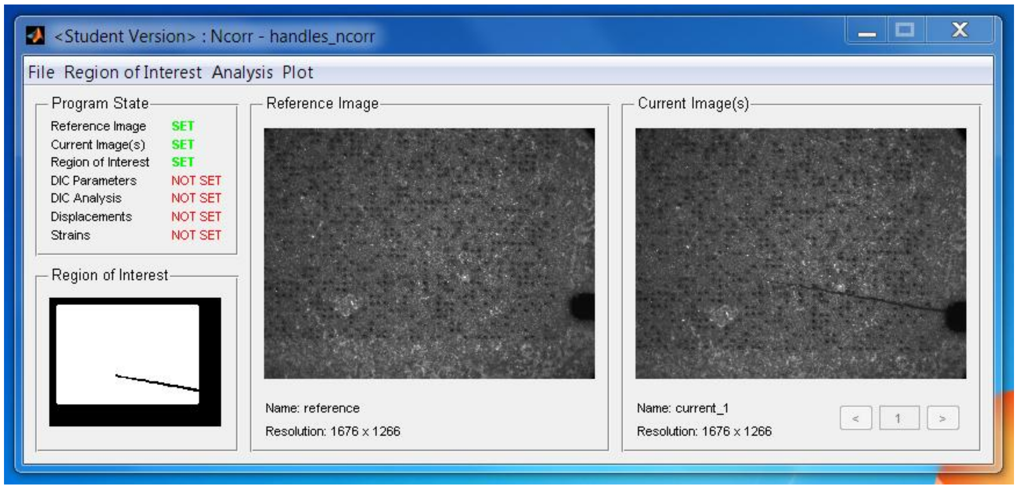
For this section, it is assumed the program has already been installed and the user has already read the General DIC section. The images used for this section (called the "cracked sample") are available off the Ncorr website if the user would like to follow along:

http://ncorr.com/download/cracksample.zip

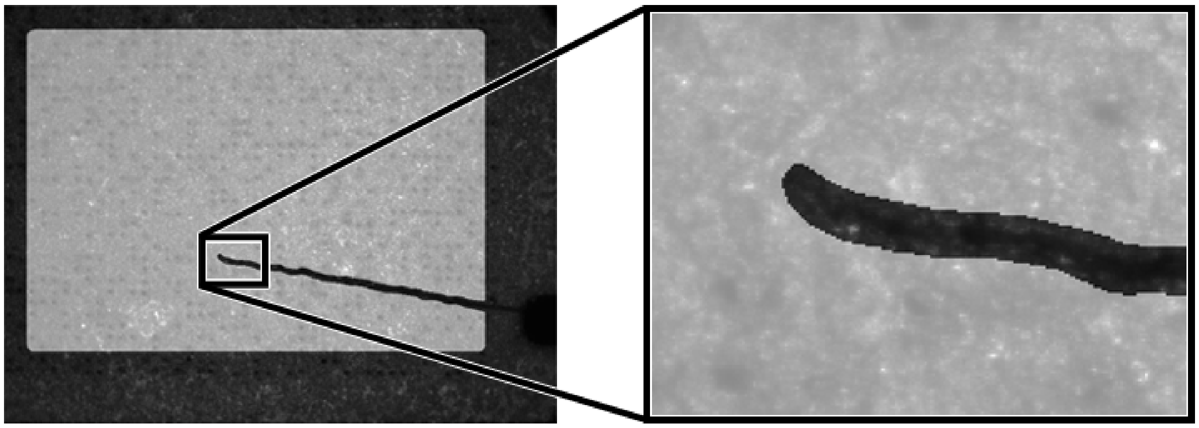
For this analysis, images from my lab are going to be used. Furthermore, to demonstrate some of the features of Ncorr, they will be loaded through the Ncorr handle instead of the GUI. The images were taken from a high resolution CCD camera and are thus quite large. To account for this, I've decided to read them through a script and then downsize them before loading them into Ncorr. The script is included with the images but is also show below:

```matlab
%% Open Ncorr
handles_ncorr = ncorr;

%% Set Parameters
% Load Ref
ref = imread('ref.TIF');
ref = imresize(ref,0.5,'bicubic','Antialiasing',true);

% Load Cur
cur = imread('cur.TIF');
cur = imresize(cur,0.5,'bicubic','Antialiasing',true);

% Load ROI
roi = rgb2gray(imread('roi_half.TIF'));
roi = roi > 5;

% Set Data
handles_ncorr.set_ref(ref);
handles_ncorr.set_cur(cur);
handles_ncorr.set_roi_cur(roi);
```

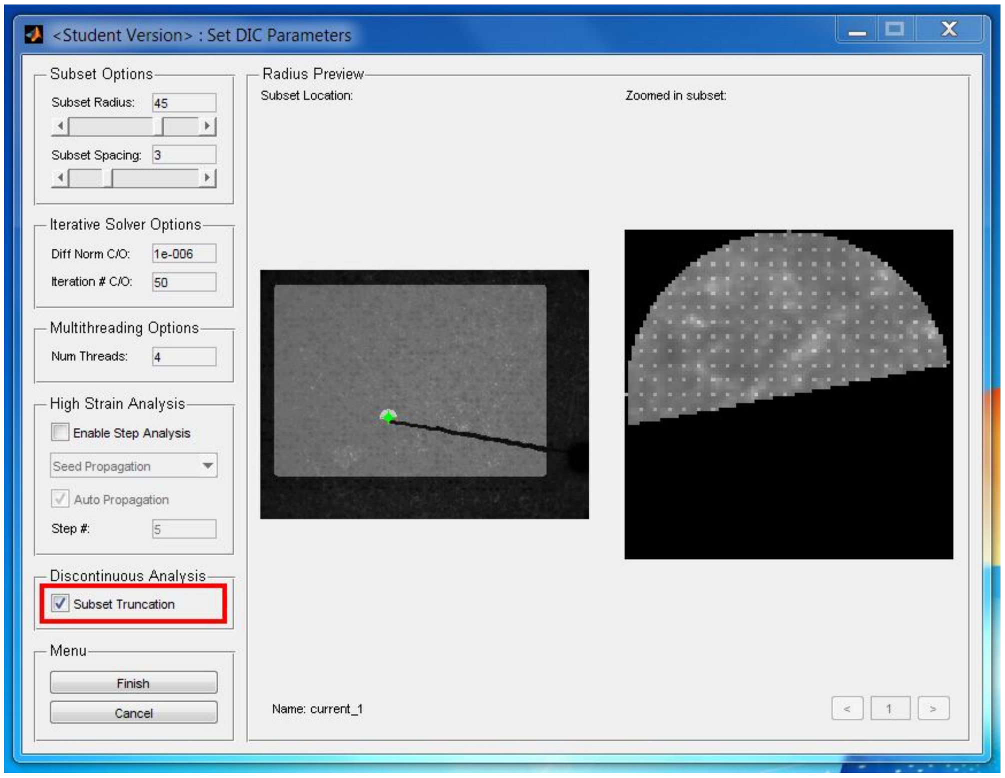
The GUI should appear as so if everything went correctly:



The important thing to note is that the ROI was set with respect to the *current* image (i.e. it uses “handles_ncorr.set_roi_cur(data)”). This is the key point to how Ncorr handles discontinuous displacement fields. The idea is that the discontinuities are clearly visible in the current configuration. The general process is that the ROI and DIC analysis proceed with respect to the current configuration, and then an Eulerian to Lagrangian algorithm converts the displacements back to the reference configuration. The ROI was traced around the crack in Photoshop; a close up of the ROI superimposed on the current image is shown below:



The DIC parameters were then set as shown below:



Subset Options
Subset Radius: 45
Subset Spacing: 3

Iterative Solver Options
Diff Norm C/O: 1e-006
Iteration # C/O: 50

Multithreading Options
Num Threads: 4

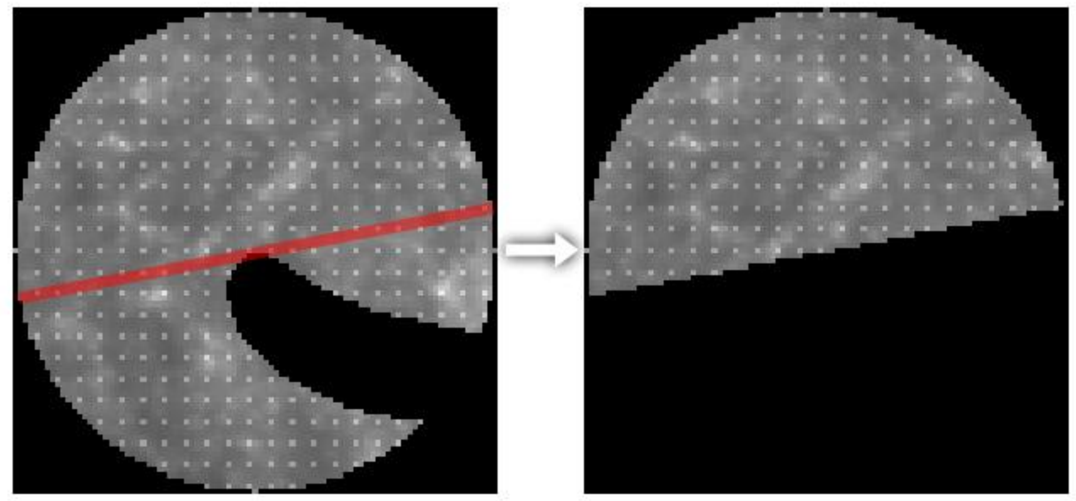
High Strain Analysis
Enable Step Analysis
Seed Propagation
[✓] Auto Propagation
Step #: 5

Discontinuous Analysis
[✓] Subset Truncation

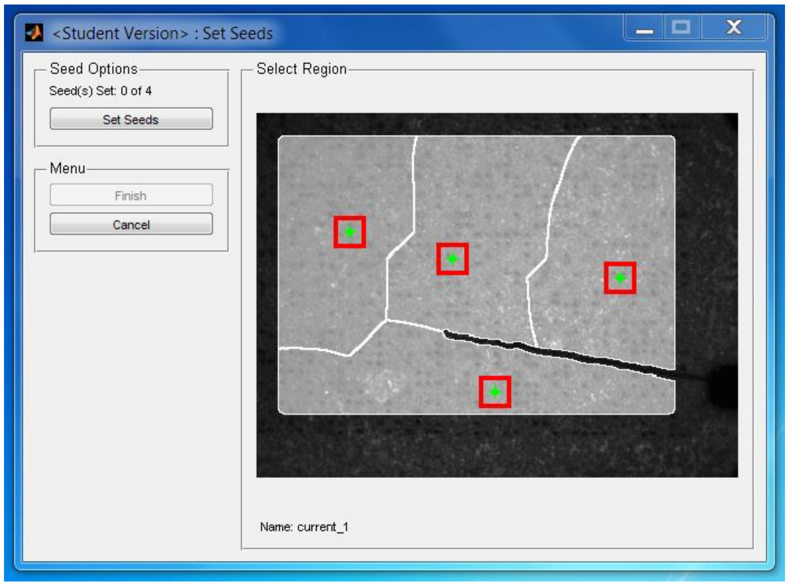
Menu

Name: current_1

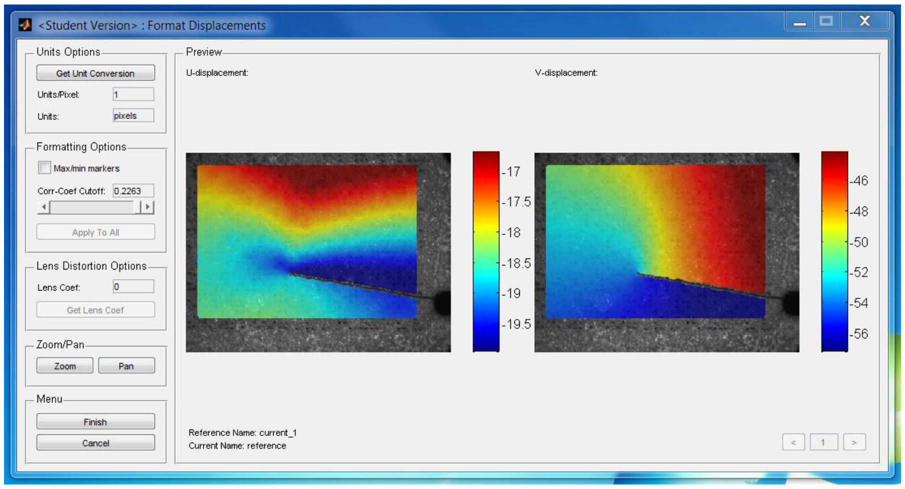
The one thing to note in particular is that subset truncation is enabled. This prevents wrap around of subsets around the crack tip. This is demonstrated below:



This is useful because the shape functions are linear, whereas the deformation near the crack tip is highly nonlinear. This helps prevent some distortions in the displacement fields which will be shown at the end of this section. Anyway, the seeds (note that if multithreading is disabled, only one seed can be placed) are placed as so:



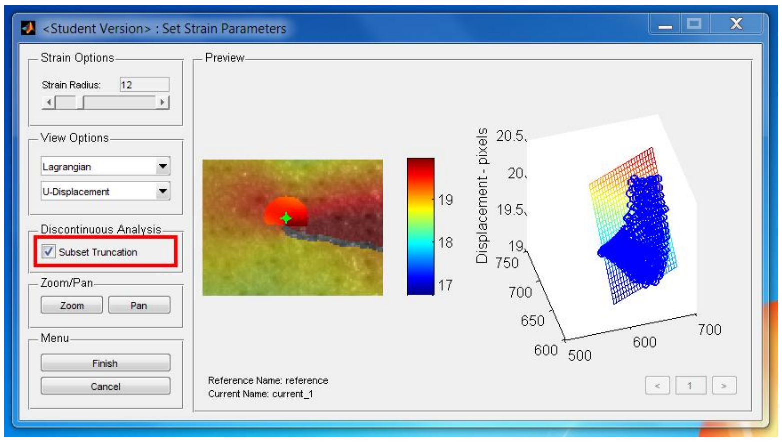
After the analysis is complete, the displacement formatting does not need any special considerations. Just accept the defaults as shown below:



Units/Pixel: 1
Units: pixels
Max/min markers: ☐
Corr-Coef Cutoff: 0.2263
Lens Coef: 0

Reference Name: current_1
Current Name: reference

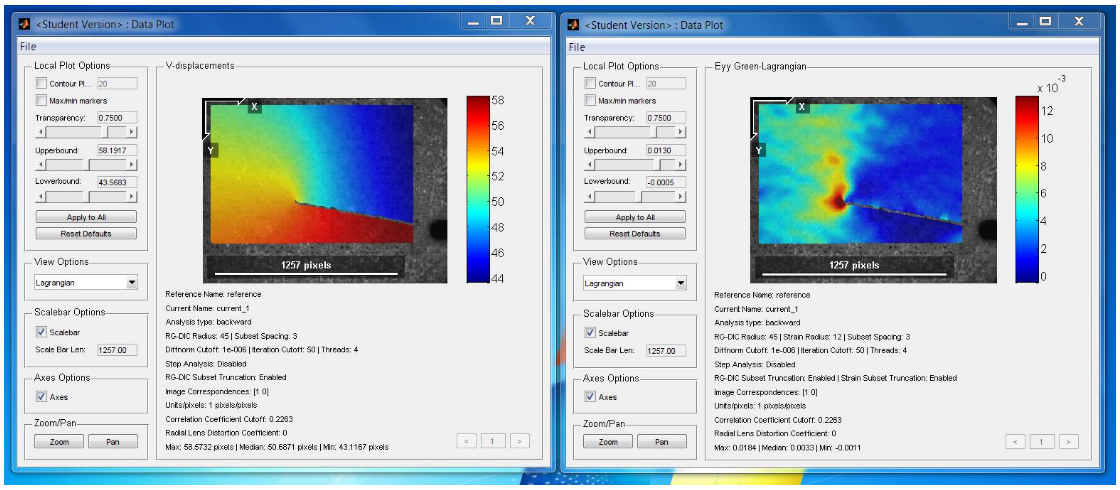
For the strain analysis, make sure to also utilize subset truncation:



Strain Radius: 12
Lagrangian
U-Displacement
☑ Subset Truncation

Reference Name: reference
Current Name: current_1

After everything is complete, the displacement and strain plots should appear as shown below (only the V displacement and E_{yy} strains are shown below since the sample is loaded in the y direction):



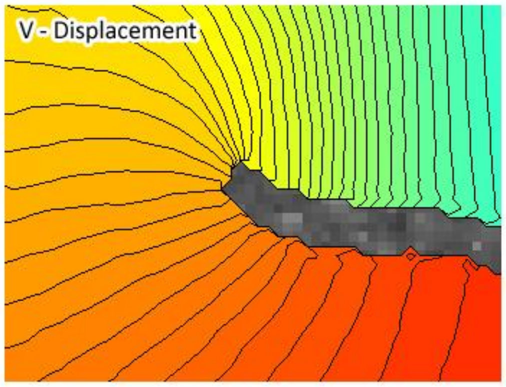
☐ Contour Pl... 20
☐ Max/min markers
Transparency: 0.7500
Upperbound: 58.1917
Lowerbound: 43.5683
Lagrangian
☑ Scalebar
Scale Bar Len: 1257.00
☑ Axes

Reference Name: reference
Current Name: current_1
Analysis type: backward
RG-DIC Radius: 45 | Subset Spacing: 3
Diffnorm Cutoff: 1e-006 | Iteration Cutoff: 50 | Threads: 4
Step Analysis: Disabled
RG-DIC Subset Truncation: Enabled
Image Correspondences: [1 0]
Units/pixels: 1 pixels/pixels
Correlation Coefficient Cutoff: 0.2263
Radial Lens Distortion Coefficient: 0
Max: 58.5732 pixels | Median: 50.6871 pixels | Min: 43.1167 pixels

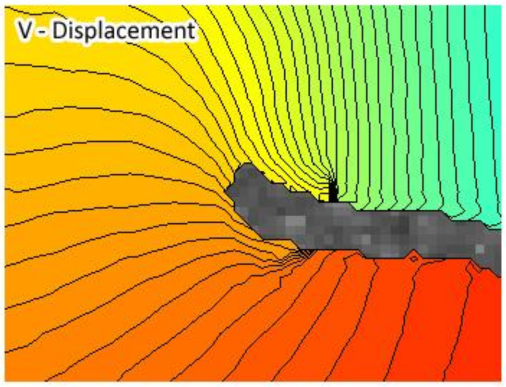
☐ Contour Pl... 20
☐ Max/min markers
Transparency: 0.7500
Upperbound: 0.0130
Lowerbound: -0.0005
Lagrangian
☑ Scalebar
Scale Bar Len: 1257.00
☑ Axes

Reference Name: reference
Current Name: current_1
Analysis type: backward
RG-DIC Radius: 45 | Strain Radius: 12 | Subset Spacing: 3
Diffnorm Cutoff: 1e-006 | Iteration Cutoff: 50 | Threads: 4
Step Analysis: Disabled
RG-DIC Subset Truncation: Enabled | Strain Subset Truncation: Enabled
Image Correspondences: [1 0]
Units/pixels: 1 pixels/pixels
Correlation Coefficient Cutoff: 0.2263
Radial Lens Distortion Coefficient: 0
Max: 0.0184 | Median: 0.0033 | Min: -0.0011

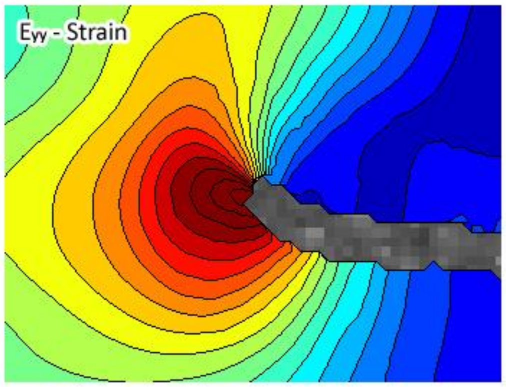
I've taken the liberty of also performing the same analysis without subset truncation. Comparisons are shown below between the zoomed-in V displacement fields (top row) and E_{yy} Lagrangian strain fields (bottom row) near the crack tip, which demonstrate the benefits of using subset truncation in mitigating distortions near the cracktip:



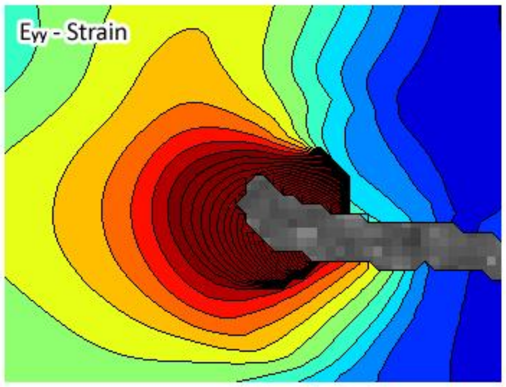
With Subset truncation



Without Subset truncation



With Subset truncation



Without Subset truncation

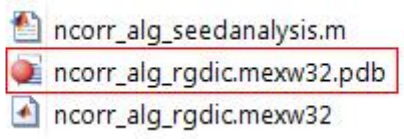
5.1 – Debugging MEX Files

This section is available for those that would like to debug the actual MEX files called by Ncorr. For example, if you are interested in examining the hessian matrix during the inverse composition Gauss Newton iterations, it would require you to debug the ncorr_alg_rgdic.cpp file. In this example, the Visual C++ 2008 IDE is used to debug this file.

The first step is to recompile the MEX file with the “-g” flag to enable debugging. For this example, we won’t be using multithreading since this making debugging easier. Type the following into the Matlab terminal:

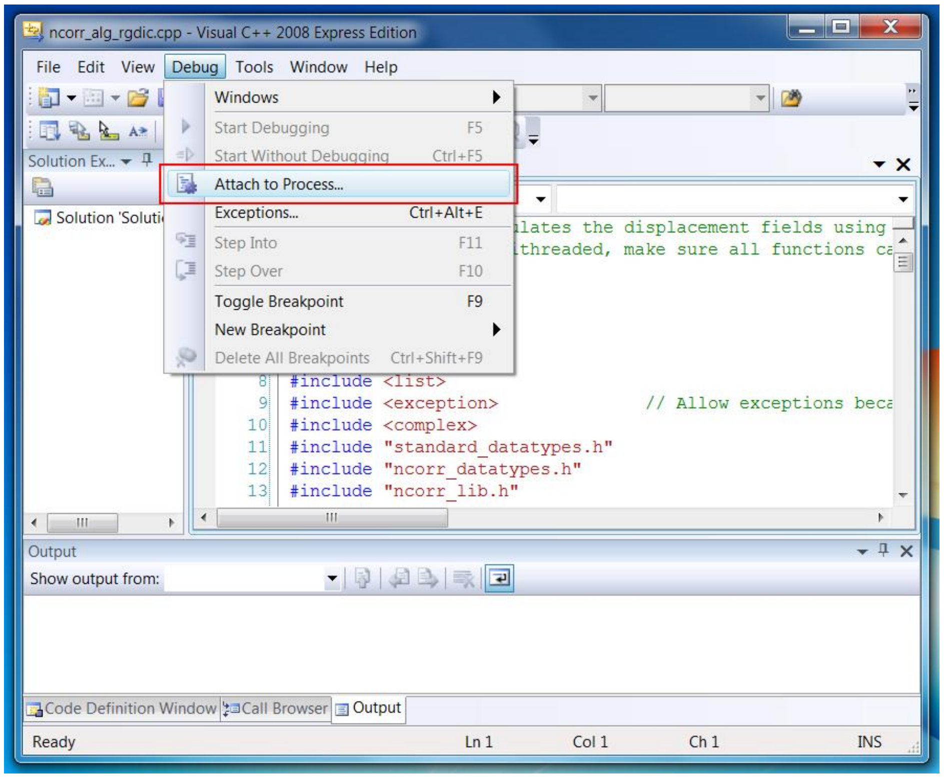
```
EDU>> mex -g ncorr_alg_rgdic.cpp standard_datatypes.obj ncorr_datatypes.obj ncorr_lib.obj
```

To ensure a debug version was successfully compiled, there should be a corresponding “.pdb” file in the directory:

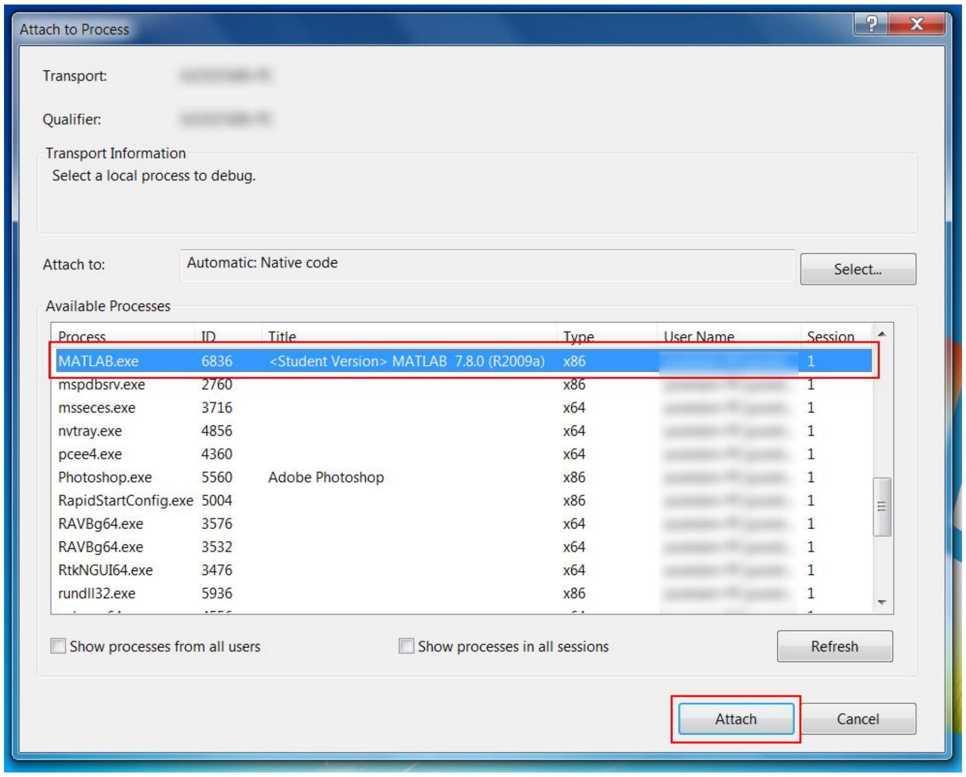


ncorr_alg_seedanalysis.m
ncorr_alg_rgdic.mexw32.pdb
ncorr_alg_rgdic.mexw32

Next, open the .cpp file in Visual Studio and attach the debugger to MATLAB.exe:



```
lates the displacement fields using
threaded, make sure all functions ca
#include <list>
#include <exception>              // Allow exceptions beca
#include <complex>
#include "standard_datatypes.h"
#include "ncorr_datatypes.h"
#include "ncorr_lib.h"
```



Attach to Process

Transport: [redacted]

Qualifier: [redacted]

Transport Information

Select a local process to debug.

Attach to: Automatic: Native code

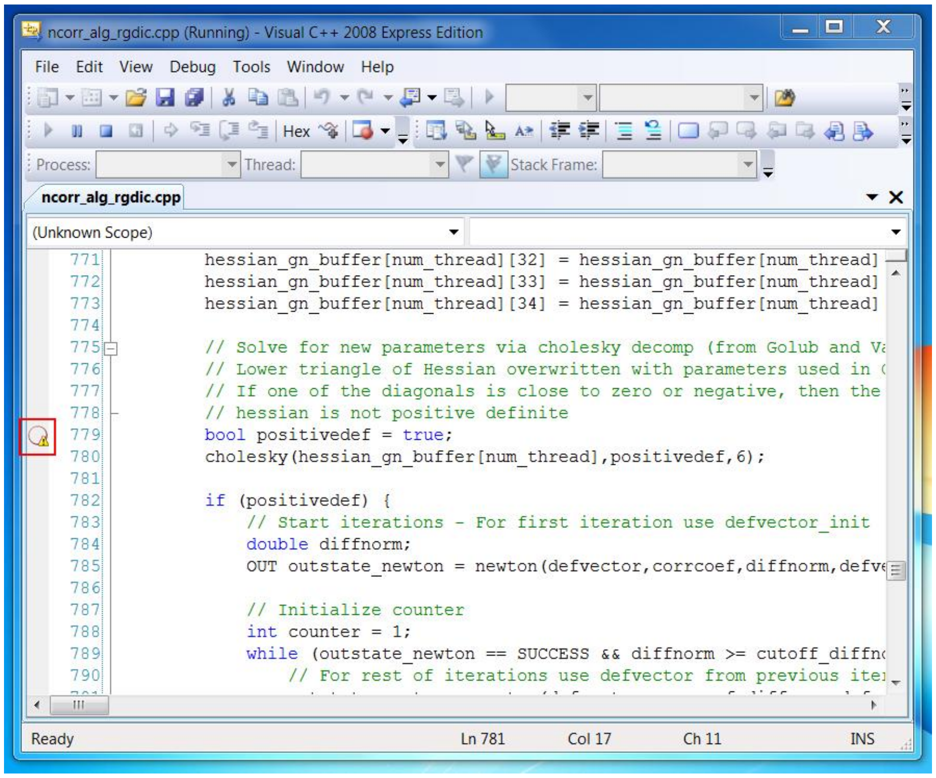
Available Processes

Process	ID	Title	Type	User Name	Session
MATLAB.exe	6836	<Student Version> MATLAB 7.8.0 (R2009a)	x86	[redacted]	1
mspdbsrv.exe	2760		x86	[redacted]	1
msseces.exe	3716		x64	[redacted]	1
nvtray.exe	4856		x64	[redacted]	1
pcee4.exe	4360		x64	[redacted]	1
Photoshop.exe	5560	Adobe Photoshop	x86	[redacted]	1
RapidStartConfig.exe	5004		x86	[redacted]	1
RAVBg64.exe	3576		x64	[redacted]	1
RAVBg64.exe	3532		x64	[redacted]	1
RtkNGUI64.exe	3476		x64	[redacted]	1
rundll32.exe	5936		x86	[redacted]	1

Show processes from all users

Show processes in all sessions

Next, set a breakpoint within the IDE and run some analysis in Ncorr in order to invoke a function call to ncorr_alg_rgdic. In this example, we are placing a breakpoint right after the hessian has been calculated as shown below:

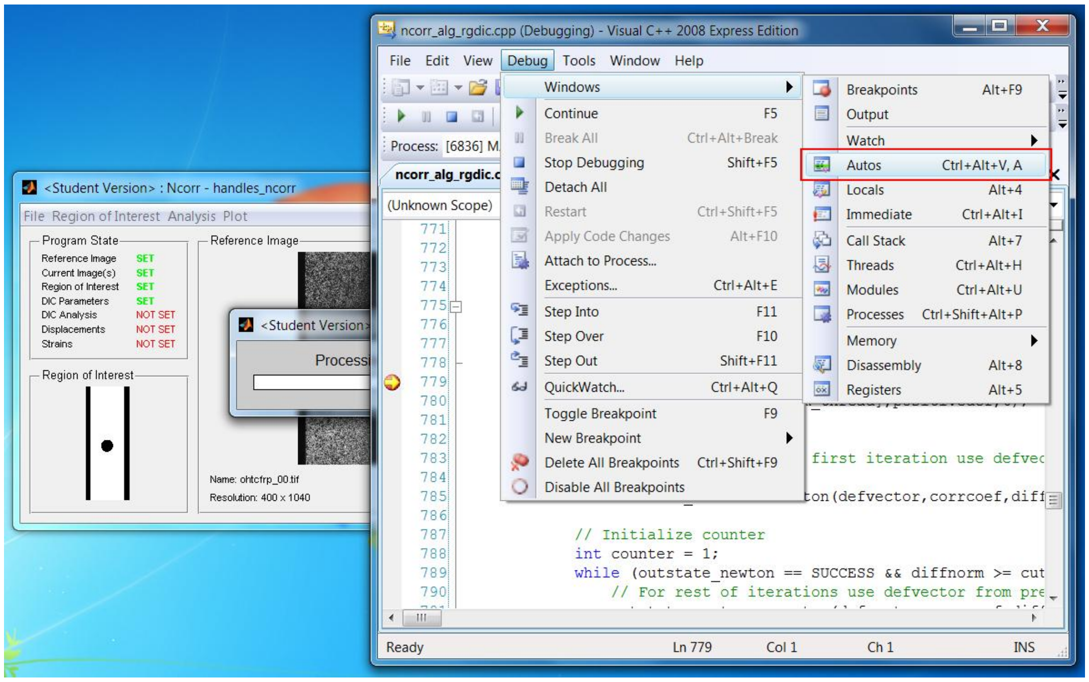


ncorr_alg_rgdic.cpp (Running) - Visual C++ 2008 Express Edition

```
        hessian_gn_buffer[num_thread][32] = hessian_gn_buffer[num_thread]
        hessian_gn_buffer[num_thread][33] = hessian_gn_buffer[num_thread]
        hessian_gn_buffer[num_thread][34] = hessian_gn_buffer[num_thread]

        // Solve for new parameters via cholesky decomp (from Golub and Va
        // Lower triangle of Hessian overwritten with parameters used in (
        // If one of the diagonals is close to zero or negative, then the
        // hessian is not positive definite
        bool positivedef = true;
        cholesky(hessian_gn_buffer[num_thread],positivedef,6);

        if (positivedef) {
            // Start iterations - For first iteration use defvector_init
            double diffnorm;
            OUT outstate_newton = newton(defvector,corrcoef,diffnorm,defve

            // Initialize counter
            int counter = 1;
            while (outstate_newton == SUCCESS && diffnorm >= cutoff_diffno
                // For rest of iterations use defvector from previous iter
```

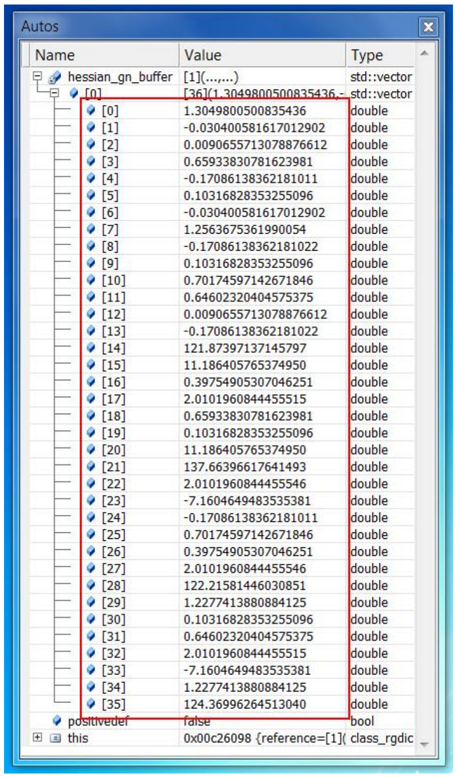
Ready Ln 781 Col 17 Ch 11 INS

After the analysis is run, the process will stop at the break point. The next step is to open a watch for the variables in the current scope:



```
first iteration use defvec
on(defvector,corrcoef,diff

    // Initialize counter
    int counter = 1;
    while (outstate_newton == SUCCESS && diffnorm >= cut
       // For rest of iterations use defvector from pre
```

This results in the following window:


Autos

Name	Value	Type
hessian_gn_buffer	[1](...,...)	std::vector
[0]	[36](1.3049800500835436,-	std::vector
[0]	1.3049800500835436	double
[1]	-0.030400581617012902	double
[2]	0.0090655713078876612	double
[3]	0.65933830781623981	double
[4]	-0.17086138362181011	double
[5]	0.10316828353255096	double
[6]	-0.030400581617012902	double
[7]	1.2563675361990054	double
[8]	-0.17086138362181022	double
[9]	0.10316828353255096	double
[10]	0.70174597142671846	double
[11]	0.64602320404575375	double
[12]	0.0090655713078876612	double
[13]	-0.17086138362181022	double
[14]	121.87397137145797	double
[15]	11.186405765374950	double
[16]	0.39754905307046251	double
[17]	2.0101960844455515	double
[18]	0.65933830781623981	double
[19]	0.10316828353255096	double
[20]	11.186405765374950	double
[21]	137.66396617641493	double
[22]	2.0101960844455546	double
[23]	-7.1604649483535381	double
[24]	-0.17086138362181011	double
[25]	0.70174597142671846	double
[26]	0.39754905307046251	double
[27]	2.0101960844455546	double
[28]	122.21581446030851	double
[29]	1.2277413880884125	double
[30]	0.10316828353255096	double
[31]	0.64602320404575375	double
[32]	2.0101960844455515	double
[33]	-7.1604649483535381	double
[34]	1.2277413880884125	double
[35]	124.36996264513040	double
positivedef	false	bool
this	0x00c26098 {reference=[1](	class_rgdic

This watch window will contain the current hessian for each thread (in this example, I have only used a single thread so there is only one hessian). Note that the hessian is a 6x6 matrix since there are 6 parameters which can be adjusted (u, v, du/dx, du/dy, dv/dx, and dv/dy). The hessian is stored in a 36 element vector using column major ordering, which is highlighted in the figure above.

With this in mind, it’s now possible to debug and check values in the MEX files called by Ncorr.