# Working with Images in MATLAB

## Teacher's Day Workshop

School of Computing and Communications

December 2013

Prepared by Dr. Abdallah Al Sabbagh in collaboration with Prof. Robin Braun and Dr. Richard Xu.

# Table of Contents

1. Introduction to MATLAB ........ 4

2. Work with Images in MATLAB ........ 5

*2.1 Read and Display an Image ........ 6*

*2.2 Grayscale Images ........ 6*

*2.3 Write the Image to a Disk File ........ 6*

*2.4 Check the Contents of the Newly Written File ........ 6*

*2.5 Resize an Image ........ 7*

*2.6 Rotate an Image ........ 8*

*2.7 Crop an Image ........ 8*

*2.8 Getting Image Pixel Values ........ 10*

*2.9 Changing Image Pixel Values ........ 11*

*2.10 Image Intensity Adjustment ........ 12*

*2.11 Detecting Edges Using the edge Function ........ 12*

*2.12 Removing Noise from an Image ........ 13*

3. Getting Help in MATLAB ........ 15

4. Alternative Softwares to MATLAB ........ 15

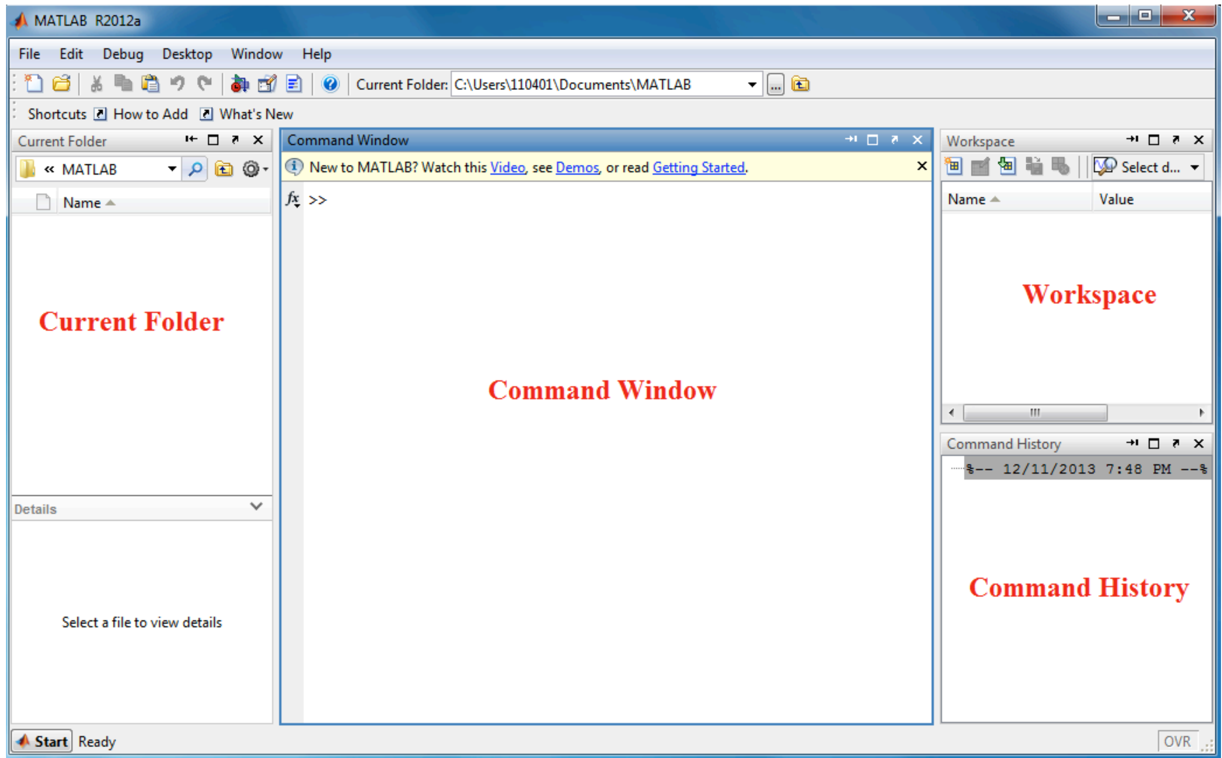
*4.1 Installing GNU Octave Software ........ 15*

5. References ........ 18

# 1. Introduction to MATLAB

MATLAB® developed by MathWorks is a high-level language and interactive environment for numerical computation, visualization, and programming.

When you start MATLAB, the desktop appears in its default layout.




The desktop includes these panels:

- **Current Folder** — Access your files.
- **Command Window** — Enter commands at the command line, indicated by the prompt (>>).
- **Workspace** — Explore data that you create or import from files.
- **Command History** — View or rerun commands that you entered at the command line.

As you work in MATLAB, you issue commands that create variables and call functions. For example, create a variable named `a` by typing this statement at the command line:

```
a=1
```

MATLAB adds variable `a` to the workspace and displays the result in the Command Window.

```
a =
     1
```

Create a few more variables.

```
b=2
    b =
        2


c=a+b
    c =
        3


d = cos(a)
    d =
        0.5403
```

Now clear all these variables from the Workspace using the *clear* command.

```
clear
```

Now clear the Command Window using the *clc* function.

```
clc
```

## 2. Work with Images in MATLAB

Digital image is composed of a two or three dimensional matrix of pixels. Individual pixels contain a number or numbers representing what grayscale or color value is assigned to it. Color pictures generally contain three times as much data as grayscale pictures, depending on what color representation scheme is used. Therefore, color pictures take three times as much computational power to process.

MATLAB can import/export several image formats:

- BMP (Microsoft Windows Bitmap)
- GIF (Graphics Interchange Files)
- HDF (Hierarchical Data Format)
- JPEG (Joint Photographic Experts Group)
- PCX (Paintbrush)
- PNG (Portable Network Graphics)
- TIFF (Tagged Image File Format)
- XWD (X Window Dump)
- raw-data and other types of image data.

## *2.1 Read and Display an Image*

You can read standard image files by using the *imread* function. The type of data returned by *imread* depends on the type of image you are reading. For example, read image1.jpg by typing (the image can be downloaded using the following link. http://crin.eng.uts.edu.au/~rob/image1.jpg, and then can be copied into the current folder):

```
A = imread('image1.jpg');
```

which will stores *image1.jpg* in a matrix named A.

Now display the image using the *imshow* function. For example, type:

```
imshow(A);
```

## *2.2 Grayscale Images*

A grayscale image is a data matrix whose values represent intensities within some range. MATLAB stores a grayscale image as an individual matrix, with each element of the matrix corresponding to one image pixel.

```
B = rgb2gray(A);
```

Now display the image by typing:

```
imshow(B);
```

## *2.3 Write the Image to a Disk File*

To write the newly adjusted image B to a disk file, use the *imwrite* function. If you include the filename extension '.png', the *imwrite* function writes the image to a file in Portable Network Graphics (PNG) format, but you can specify other formats. For example, type:

```
imwrite (B, 'image2.png');
```

## *2.4 Check the Contents of the Newly Written File*

To see what *imwrite* wrote to the disk file, use the *imfinfo* function.

```
imfinfo('image2.png')
```

The *imfinfo* function returns information about the image in the file, such as its format, size, width, and height.

```
ans =

                 Filename: 'image2.png'
              FileModDate: '12-Nov-2013 10:43:31'
                 FileSize: 52936
                   Format: 'png'
            FormatVersion: []
                    Width: 350
                   Height: 350
                 BitDepth: 8
                ColorType: 'grayscale'
          FormatSignature: [137 80 78 71 13 10 26 10]
                 Colormap: []
                Histogram: []
            InterlaceType: 'none'
             Transparency: 'none'
   SimpleTransparencyData: []
          BackgroundColor: []
          RenderingIntent: []
           Chromaticities: []
                    Gamma: []
              XResolution: []
              YResolution: []
           ResolutionUnit: []
                  XOffset: []
                  YOffset: []
               OffsetUnit: []
          SignificantBits: []
             ImageModTime: '11 Nov 2013 23:43:31 +0000'
                    Title: []
                   Author: []
              Description: []
                Copyright: []
             CreationTime: []
                 Software: []
               Disclaimer: []
                  Warning: []
                   Source: []
                  Comment: []
                OtherText: []
```

## *2.5 Resize an Image*

To resize an image, use the `imresize` function. When you resize an image, you specify the image to be resized and the magnification factor. To enlarge an image, specify a magnification factor greater than 1. To reduce an image, specify a magnification factor between 0 and 1.

```
imshow(B);

C = imresize(B,1.5);
```

```
figure
imshow(C);

C = imresize(B,0.5);
figure
imshow(C);
```

You can specify the size of the output image by passing a vector that contains the number of rows and columns in the output image. If the specified size does not produce the same aspect ratio as the input image, the output image will be distorted.

```
C = imresize(B,[300,150]);
figure
imshow(C);
```

This example creates an output image with 300 rows and 150 columns.

## *2.6 Rotate an Image*

To rotate an image, use the `imrotate` function. When you rotate an image, you specify the image to be rotated and the rotation angle, in degrees. If you specify a positive rotation angle, `imrotate` rotates the image counterclockwise; if you specify a negative rotation angle, `imrotate` rotates the image clockwise.

```
C = imrotate(B,35);
figure
imshow(C);

C = imrotate(B,-20);
figure
imshow(C);
```

## *2.7 Crop an Image*

Cropping an image means creating a new image from a part of an original image. To crop an image using the Image Viewer, use the ***Crop Image*** tool or use the `imcrop` function.

**Using the Crop Image Tool:**

By default, if you close the Image Viewer, it does not save the modified image data. To save the cropped image, you can use the Save As option from the Image Viewer File menu to store

the modified data in a file or use the Export to Workspace option to save the modified data in the workspace variable. To use the Crop Image tool, follow this procedure:

**1)** View an image in the Image Viewer.

```
imtool(A);
```

**2)** Start the Crop Image tool by clicking ***Crop Image*** in the Image Viewer toolbar or selecting ***Crop Image*** from the Image Viewer Tools menu. (Another option is to open a figure window with `imshow` and call `imcrop` from the command line.) When you move the pointer over the image, the pointer changes to cross hairs .

**3)** Define the rectangular crop region, by clicking and dragging the mouse over the image. You can fine-tune the crop rectangle by moving and resizing the crop rectangle using the mouse.

**4)** When you are finished defining the crop region, perform the crop operation. Double-click the left mouse button or right-click inside the region and select ***Crop Image*** from the context menu. The Image Viewer displays the cropped image.

**5)** To save the cropped image, use the Save as option or the Export to Workspace option on the Image Viewer File menu.

Now display the image using the *`imshow`* function.

**<u>Using the imcrop Function:</u>**

By using the *`imcrop`* function, you can specify the crop region interactively using the mouse or programmatically by specifying the size and position of the crop region.

This example illustrates an interactive syntax. The example reads an image into the MATLAB workspace and calls *`imcrop`* specifying the image as an argument. *`imcrop`* displays the image in a figure window and waits for you to draw the crop rectangle on the image. When you move the pointer over the image, the shape of the pointer changes to cross hairs . Click and drag the pointer to specify the size and position of the crop rectangle. You can move and adjust the size of the crop rectangle using the mouse. When you are satisfied with the crop rectangle, double-click to perform the crop operation, or right-click inside the crop rectangle and select Crop Image from the context menu. *`imcrop`* returns the cropped image.

```
C = imcrop(A);

figure

imshow(C);
```

**Raw MATLAB:** For advanced users, the native MATLAB commands can be used. You can specify the size and position of the crop rectangle as parameters when you call *imcrop*. Specify the crop rectangle as a four-element position vector, [xmin ymin width height].

In this example, you call *imcrop* specifying the image to crop, A, and the crop rectangle. *imcrop* returns the cropped image in D.

```
D = imcrop(A,[160 140 110 180]);
figure
imshow(D);
```

## *2.8 Getting Image Pixel Values*

You can get information about specific image pixels such as RGB values. Type:

```
A(2,15,:)
```

which returns the RGB (red, green, and blue) color values of the pixel (2,15). R=66; G= 88; B= 174.

```
ans(:,:,1) =

    66

ans(:,:,2) =

    88

ans(:,:,3) =

   174
```

Now try:

```
A(40:100,10:20,:)
```

You can also use the *impixel* function which will determine the values of one or more pixels in an image and return the values in a variable. Select the pixels interactively using a mouse. *impixel* returns the value of specified pixels in a variable in the MATLAB workspace.

The following example illustrates how to use *impixel* to get pixel values.

**1)** Display an image.

```
imshow(A);
```

**2)** Call *impixel*. When called with no input arguments, *impixel* associates itself with the image in the current axes.

```
vals = impixel
```

**3)** Select the points you want to examine in the image by clicking the mouse. `impixel` places a star at each point you select.

**4)** When you are finished selecting points, press Return. `impixel` returns the pixel values in an n-by-3 array, where n is the number of points you selected. The stars used to indicate selected points disappear from the image.

```
vals =

    46    71   155
    80    96   184
    95   107   193
```

## *2.9 Changing Image Pixel Values*

You can change the values of specific image pixels. Type:

```
A(40:100,10:20,:) = 0;
figure
imshow(A);
```

which changes the colors of the selected pixels into black color.

Now try:

```
A(40:100,10:20,:) = 255;
```

```
figure
imshow(A);
```

## *2.10 Image Intensity Adjustment*

Image intensity adjustment is used to improve an image, Read *image1.jpg* again.

```
A = imread('image1.jpg');
```

Multiply the image pixels values by two.

```
E = A.*2;
figure
imshow(E);
```

Now try:

```
F = A.*0.75;
figure
imshow(F);
```

Then, try:

```
F = A.*7.5;
figure
imshow(F);
```

## *2.11 Detecting Edges Using the edge Function*

In an image, an edge is a curve that follows a path of rapid change in image intensity. Edges are often associated with the boundaries of objects in a scene. Edge detection is used to identify the edges in an image. To find edges, you can use the `edge` function. This function looks for places in the image where the intensity changes rapidly, using one of these two criteria:

- Places where the first derivative of the intensity is larger in magnitude than some threshold.
- Places where the second derivative of the intensity has a zero crossing.

`edge` provides a number of derivative estimators, each of which implements one of the definitions above. For some of these estimators, you can specify whether the operation should be sensitive to horizontal edges, vertical edges, or both. edge returns a binary image containing 1's where edges are found and 0's elsewhere.

The most powerful edge-detection method that edge provides is the `Canny` method. The Canny method differs from the other edge-detection methods in that it uses two different thresholds (to detect strong and weak edges), and includes the weak edges in the output only if they are connected to strong edges. This method is therefore less likely than the others to be fooled by noise, and more likely to detect true weak edges.

The following example illustrates the power of the `Canny` edge detector by showing the results of applying the `Sobel` and `Canny` edge detectors to the same image:

**1)** Read the image and display it.

```
G = imread('image2.png');

imshow(G);
```

**2)** Apply the `Sobel` and `Canny` edge detectors to the image and display them.

```
BW1 = edge(G,'sobel');

BW2 = edge(G,'canny');


figure

imshow(BW1);


figure

imshow(BW2);
```

## *2.12 Removing Noise from an Image*

Digital images are prone to a variety of types of noise. Noise is the result of errors in the image acquisition process that result in pixel values that do not reflect the true intensities of the real scene. There are several ways that noise can be introduced into an image, depending on how the image is created. For example:

- If the image is scanned from a photograph made on film, the film grain is a source of noise. Noise can also be the result of damage to the film, or be introduced by the scanner itself.
- If the image is acquired directly in a digital format, the mechanism for gathering the data (such as a CCD detector) can introduce noise.
- Electronic transmission of image data can introduce noise.

You can use linear filtering to remove certain types of noise. Certain filters, such as averaging or Gaussian filters, are appropriate for this purpose. For example, an averaging filter is useful for removing grain noise from a photograph. Because each pixel gets set to the average of the pixels in its neighborhood, local variations caused by grain are reduced.

Median filtering is similar to using an averaging filter, in that each output pixel is set to an average of the pixel values in the neighborhood of the corresponding input pixel. However, with median filtering, the value of an output pixel is determined by the median of the neighborhood pixels, rather than the mean. The median is much less sensitive than the mean to extreme values (called outliers). Median filtering is therefore better able to remove these outliers without reducing the sharpness of the image.

The following example compares the use of a linear Gaussian filter and a median filter to remove salt and pepper noise for the same image:

**1)** Read the image and display it.

```
H = imread('image2.png');

imshow(H);
```

**2)** Add salt and pepper noise to the image and then display it.

```
I = imnoise(H,'salt & pepper',0.02);


figure

imshow(I);
```

**3)** Filter the noisy image using a linear Gaussian filter.

- Create a Gaussian filter using the `fspecial` function.

  ```
  filter = fspecial('gaussian',[3 3], 0.5);
  ```

- Filter the image using the created filter and then display the filtered image.

  ```
  J = imfilter(I, filter, 'replicate');


  figure

  imshow(J);
  ```

**4)** Filter the noisy image using a median filter by applying the `medfilt2` function and then display the filtered image.

```
K = medfilt2(I,[3 3]);


figure

imshow(K);
```

# 3. Getting Help in MATLAB

For reference information about any of the functions, type in the MATLAB command window:

```
help functionname
```

For example:

```
help imread
```

# 4. Alternative Softwares to MATLAB

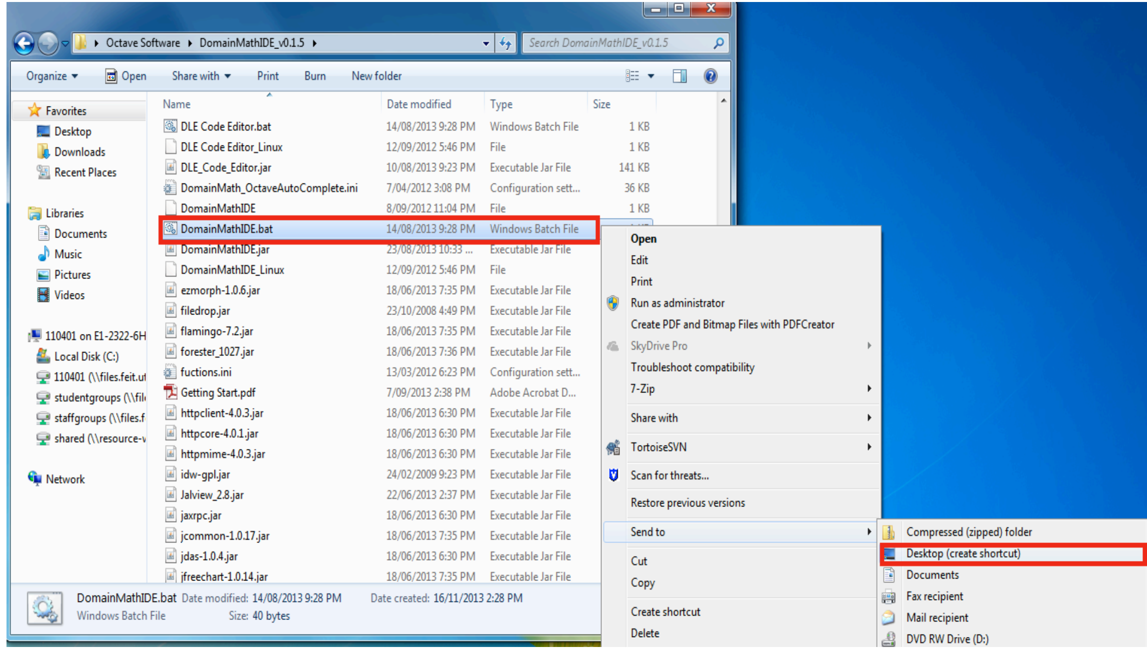
There are several open source alternatives softwares to MATLAB such as: GNU Octave, FreeMat, and Scilab. GNU Octave will be installed in this section.

## *4.1 Installing GNU Octave Software*

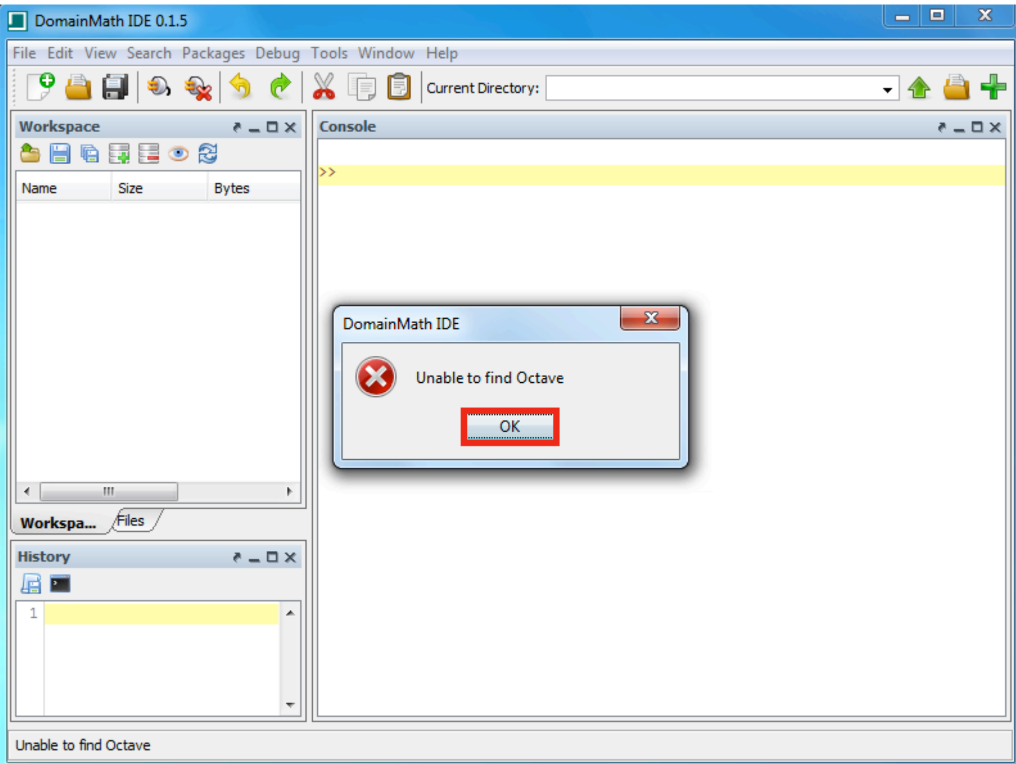
**1)** Download both *"Octave.zip"* and *"DomainMathIDE.zip"* zip files. Then, unpack these two zip files from the following links:

a. http://crin.eng.uts.edu.au/~rob/DomainMathIDE.zip
b. http://crin.eng.uts.edu.au/~rob/Octave.zip

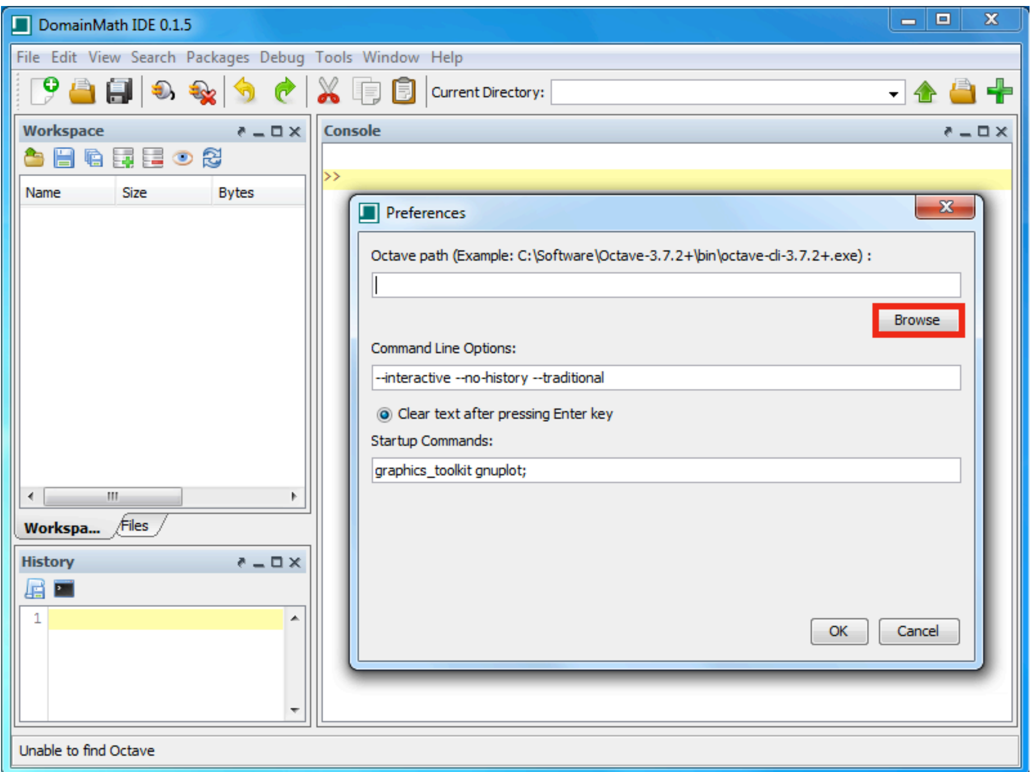
**2)** From the *DomainMathIDE_v0.1.5* directory, create a shortcut for the *"DomainMathIDE.bat"* file.

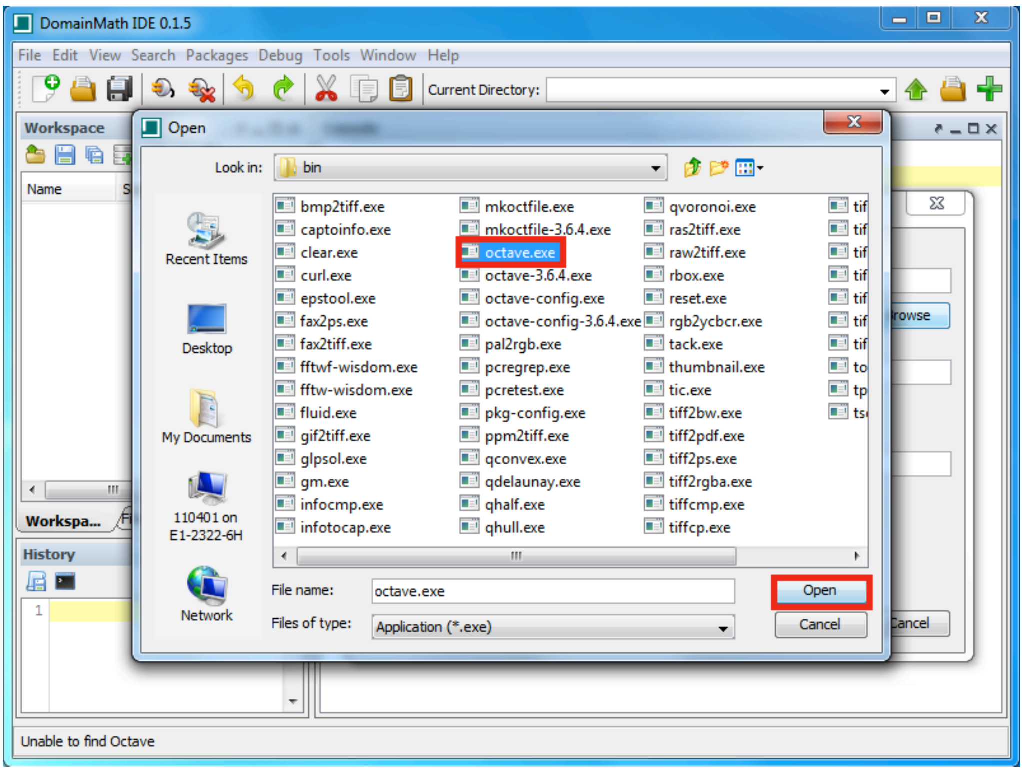


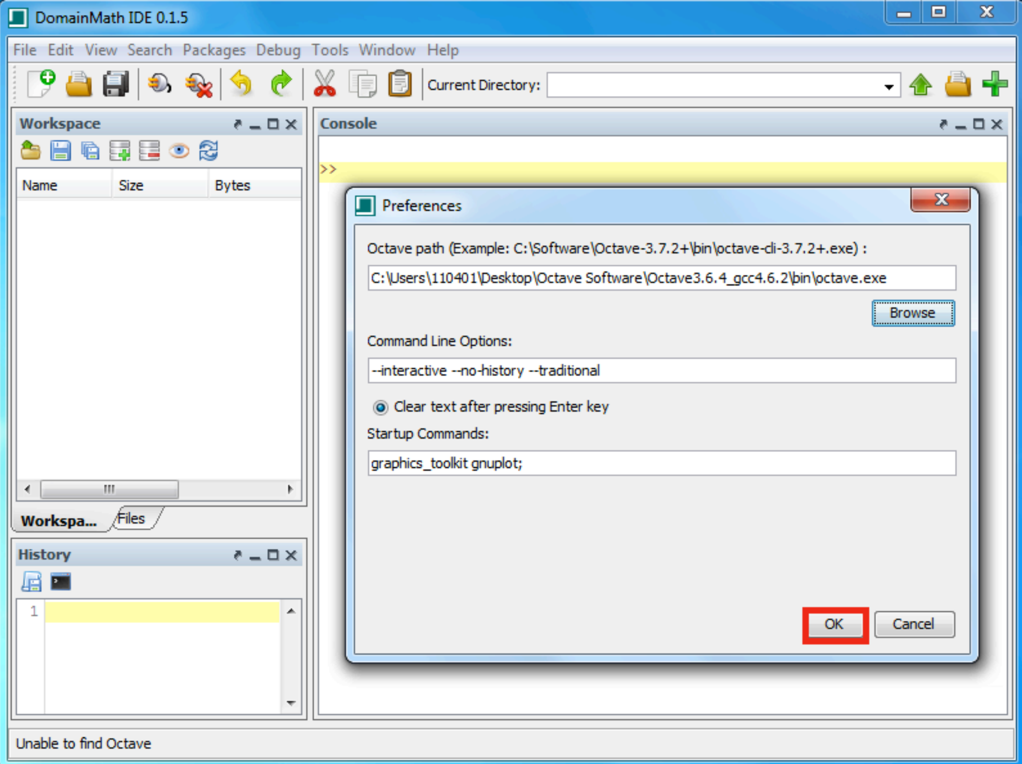
**3)** From the *Desktop*, open the created shortcut. You will get a message: "Unable to find Octave", click OK and then add Octave path.



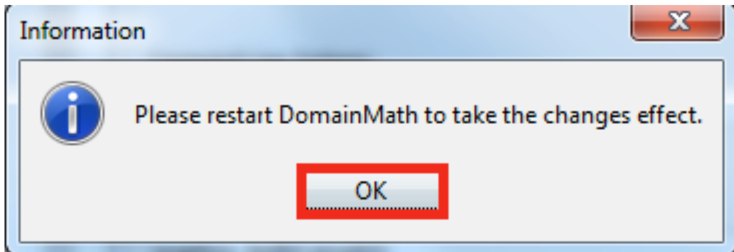

**4)** Add the Octave path; browse the *octave.exe* file from the *bin* directory inside the *Octave3.6.4_gcc4.6.2* directory.



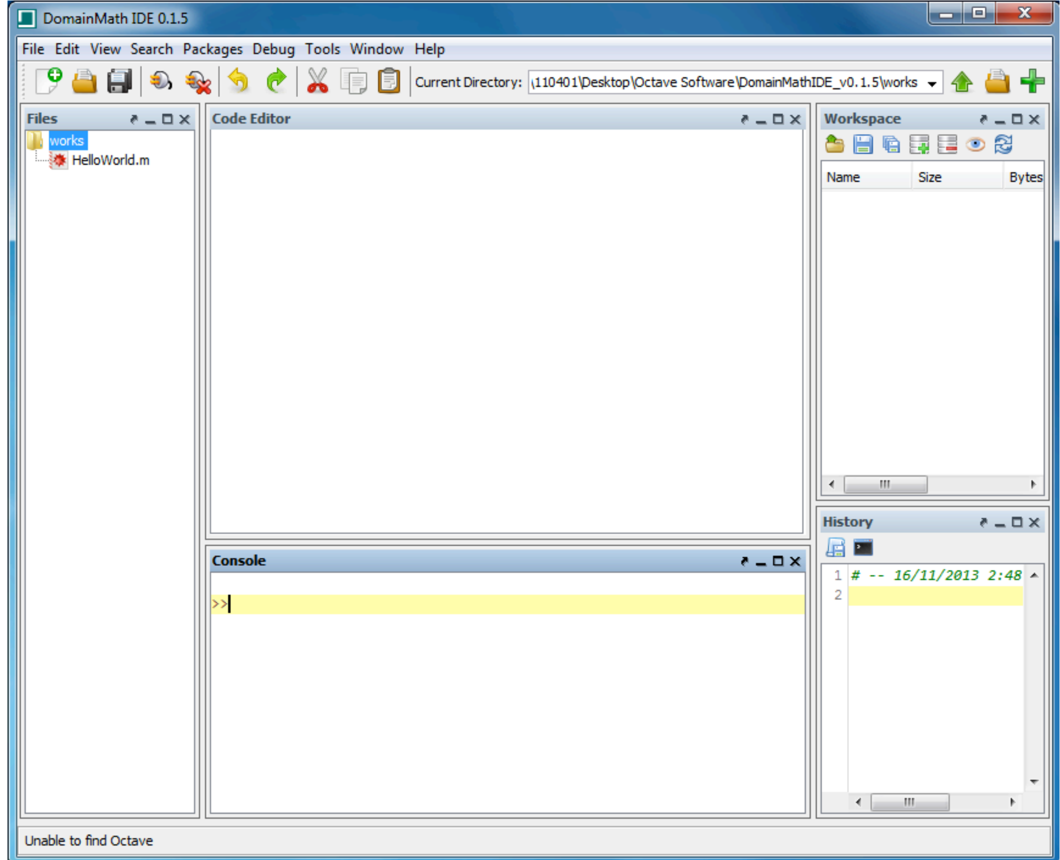



**5)** Click OK to restart the *DomainMath* application which it is an open source GUI front-end application for GNU Octave.




**6)** Now type the command inside the console window.




## 5. References


MathWorks, *MATLAB® Primer*, The MathWorks, Natick, MA, USA 2013.

MathWorks, *Image Processing Toolbox™ User's Guide*, The MathWorks, Natick, MA, USA 2013.

A. Knott, "MATLAB 6.5 Image Processing Toolbox Tutorial," Department of Computer Science, University of Otago, Dunedin, New Zealand.