22481 Introduction to medical imaging

Guide to the SIS toolbox

Jens E. Wilhjelm, Line Marcussen & Anders Emil H. Jakobsen

Biomedical Instrumentation, Health Technology, Ørsteds Plads, Building 349

Technical University of Denmark

DK-2800 Kgs. Lyngby

(Ver. 4.4 6/9/21) © 2005-2021 by J. E. Wilhjelm

1 Introduction

A toolbox is a collection of functions that can be used for a special purpose. The `SIS` toolbox is meant for importing, analyzing, manipulating and visualizing *metric* medical images.

This tutorial is meant to be a self-study tutorial for the relatively experienced MATLAB user who needs to use the `SIS` toolbox in the course 22481 (previously 31540) Introduction to medical imaging.

If you have this document open, you can cut and paste the different commands directly in to MATLAB.

2 Accessing the necessary toolboxes

There are actually two toolboxes: The SIS toolbox with the functions that you need and additional toolbox, `sis_tools`, with functions that the SIS toolbox need.

The toolboxes that you need are on the Web. Do the following to install them:

- Go to: courses.healthtech.dtu.dk/22481/?Programs/main.html
- For both toolboxes, download `complete.zip` to your network drive (or other personal drive with your own files)
- Unzip the `complete.zip` file (winzip, pkzip, etc).
- Place the files in a folder where you can find them later (e.g. course specific folder or folder for MATLAB toolboxes). Each toolbox must be in its own directory.

Remember that updates might become available during the course requiring you to redo this process.

Create a startup file (`startup.m`) with paths to the `SIS` and `sis_tools` toolboxes:

```
path( path, 'PathToSISToolBox');,

path( path, 'PathToSIS_toolsToolBox');,
```

Here, `PathToSISToolBox` and `PathToSIS_toolsToolBox` must be replaced with valid paths. This file must be executed at the beginning of each MATLAB session. For automatic execution, a file called startup.m can be added to your default directory with the given commands. For details see http://se.mathworks.com/help/matlab/ref/startup.html.

Do not use time on consulting the `sis_tools` toolboxe, it only contains internal routines for `sis`.

You can always execute `startup.m` manually, by typing `startup`. Write `path` and check that you have a path to all relevant toolboxes. Try also `help sis` (there is no help for `sis_tools`). (`help sis` only works, if you sis directory is called “sis”, e.g.: c:\users\rasmus\matlab_tools\sis.)

3 Which MATLAB is needed?

It is difficult to state exactly which MATLAB version you need, but versions 2017a and above will probably work. However, it is mandatory to have the *Image Processing Toolbox* from MATLAB (called `images` for short), since SIS uses this toolbox intensively.

You should run the script `sis_welcome` to verify the presence of various toolboxes and see memory status.

4 Part 1 (without SIS)

In both tutorials in this document, only synthetic data generated by a random generator will be used. However, the real power of the SIS toolbox comes about when real medical images are to be loaded, as will be seen a little later in the course.

4.1 Generation of images

Try to make a random image by typing `MyImage = randn(50, 70);`. Check the size with *e.g.* `whos`. Check the range of the image values, *e.g.* `min(min(MyImage))`[1]. Why do we use the function `min` two times? Likewise for the maximum values. Try to "normalize" `MyImage` so that it has values between 0 and 1.

4.2 Plotting images

You can plot your image with `imagesc(MyImage);`. To display the range of `MyImage`, type `colorbar;` afterwards. Try also `colormap gray`. What does the colors show?

What are the numbers on the two axes telling you? If you know the physical distance between two neighboring pixels, you can use this to create the correct horizontal and vertical axis. Let’s say that this distance is 10 mm in both directions in the image. Then the axis vectors can be written as:

```
horiz_axis = (0:69)*10;
vert_axis = (0:49)*10;
```

Write `whos` to see the workspace. You can now plot the image with the correct axis by:

```
imagesc( horiz_axis, vert_axis, MyImage); colorbar;
xlabel('Horizontal (mm)');  ylabel('Vertical (mm)');
```

Notices that the axis with length 70 is given *first*, while `70` was written as the *second* argument in `MyImage = randn(50, 70);`. This is because the "70" is the number of *columns* in `MyImage` but when plotted with `imagesc`, it corresponds to the *horizontal* axis (the "x-axis").

4.3 Creating multidimensional images

How would you create multiple images? Well, one way would be to write `MyImage = rand(50, 70, 4)`, so that `MyImage` becomes a 3D array that can be considered as four images, each 50 pixels high and 70 pixels wide. Assume that the spacing is 1 mm between pixels in the image and 10 mm between images. Please create the associated three axis. Now how to plot these, without too much trouble? Well, this is what the second tutorial is about.

1. You could also write `min( MyImages(:) )`. The reason is that while `MyImages` is an *array*, `MyImages(:)` is a *vector*.

5 Part 2 (using SIS and synthetic data)

You will now use the SIS toolbox and the toolboxe(s) needed for the SIS toolbox.

Look at the help for the SIS toolbox, to get an idea of the names of the functions, or at least the groups of functions that are present. Simply write `help sis`.

5.1 Creating data in SIS format

We will now create a SIS structure based on the above data from the last step of Tutorial part 1. You can put all your commands - with some helpful comments about the functionality - in a script (an m-file). First of all you need the image itself. Since we will call the entire structure `Data`, this is simply:

```
close all; clear all;
Data.Images = randn(50, 70, 4);
Data.ImageType = 'intensity';
```

where the data is just random numbers. We here consider the data as “intensity” data. Please consider Appendix 10.1 to see the different types of data allowed or ignore `Data.ImageType` for now.

Next you will need to generate a metric axis that can accompany this image. Since the image is synthetic, any axis will do. The first one is:

```
Data.Axes(1).Axis = (0:49); % 0 1 2 3 ... mm
Data.Axes(1).Label = 'Vertical axis';
Data.Axes(1).Symbol = 'y';
Data.Axes(1).Unit = 'mm';
```

The second one could be

```
Data.Axes(2).Axis = (0:69)-25; % -25 -24 ... mm
Data.Axes(2).Label = 'Horizontal axis';
Data.Axes(2).Symbol = 'x';
Data.Axes(2).Unit = 'mm';
```

and the third one could finally be:

```
Data.Axes(3).Axis = (0:3)*10; % 0 10 20 30 mm
Data.Axes(3).Label = 'Depth axis';
Data.Axes(3).Symbol = 'z';
Data.Axes(3).Unit = 'mm';
```

With these three axes, what is the distance between two voxels[1] in all three spatial dimensions? Are there any difference?

After having created the axes of the image, next some information about this image itself is needed; This could be:

```
Data.ImagesLabel = 'Magnitude';
Data.ImagesSymbol = '|g_r(t)|';
Data.ImagesUnit = 'V';
```

1. Images consist of *pixels* (picture elements), volume images (3D images) consist of *voxels* (volume elements).

The contents of three lines above is just some text that does not give so much meaning without the original data being more formally described (so do not think more about this, in this tutorial). The last fields that we need are:

```
Data.Date = now; % type "help now" in MATLAB for information on format
Data.Object = 'snow (random numbers)';
Data.Operator = 'Your name';
Data.Where = 'in my computer';
Data.ScannerType = 'Simulation';
Data.ScannerId = 'Matlab';
Data.Settings = []; % just empty, but can contain specific info about scanner
Data.SisVersion = 2;
```

When the data structure is finished it can be plotted with

```
sis_view( Data);
```

It can happen that you get an error message, like `??? Reference to non-existent field`. See if that field is missing. Another way of debugging is to use

```
sis_consistency_check(Data)
```

If things fails no matter what, and you cannot get `sis_view` to plot, then use `sis_function` directly to create a SIS structure. Compare this with your own data to see if there is a difference that could explain the error.

5.2 Zooming on data

In the previous subsection, a figure with four images were generated. Now, a part of a single image will be extracted by use of `sis_zoom`. Assume that a central part of the second image is to be extracted. This is done by:

```
Data_zoom = sis_zoom( Data, [10 0 2], [35 40 2], 'mmi');
```

where `[10 0 2]` is the starting point in 3D (thus three coordinates) and `[35 40 2]` is the corresponding ending point in 3D. `'mmi'` indicates that the first two numbers should be interpreted as metric values, while the last is simple index (which can be 1, 2, 3 or 4 since there are four images). So `'m'` = metric, while `'i'` = indices. From the above call, you can see, that `sis_zoom` can zoom in all dimensions in one single call. If you want a dimension unaltered, then `help sis_zoom` tells you how. Note that both image and axes are zoomed on.

Now, make a new figure and plot this

```
figure;
sis_view( Data_zoom);
```

Be sure to check that the axes on both figures matches those in the call to `sis_zoom`. You will use `sis_zoom` intensively in the course.

If you want to zoom on a particular image (when viewing the images) you can use the zoom tool on the menu bar of the figure, but then your data will not be changed.

5.3 Plotting lines

Before you continue, delete the first figure, so that you only have the last figure left with only one image.

In order to see some of the possibilities of Matlab and SIS, do the following: When `sis_view` has been executed to make a plot, then use `hold on` followed by the use of the `plot` command. Try to make 5 horizontal, equally-spaced red lines across the image. Example: `plot( [0 40],[15 15], 'r');`. Then use `hold off`.

5.4 Changing zero point of axis

Now, go back to the image created in Subsection 5.2. Change the axis, so that the zero point of the axes is at the centre of the image. You can do the following. Make a copy of `Data_zoom`:

```
Data_z = Data_zoom;
```

Then change the first axis, so that the zero point is at the centre:

```
Data_z.Axes(1).Axis = Data_z.Axes(1).Axis - mean(Data_z.Axes(1).Axis);
```

Then plot

```
figure;
sis_view( Data_z);
```

Did the zero point change as expected? How would you handle the other dimensions?

5.5 Miscellaneous

When the program works, check that you have made clear and concise comments to each line, and that you have made an appropriate header in the file. The header should in the first line contain `% Script FILENAME` and in the next lines there should be a description of what the program does. The last line of the header should contain version number, date, place, your name, etc.

6 Part 3 (using SIS and MATLAB's MRI data)

You will now use the SIS toolbox on a 3D diffusion-weighted MR image of the brain. This image is only used for demonstrating some of the functions of the SIS toolbox. To get an overview of the functions in the toolbox, type `help sis.` It is recommended that you put all the commands (with comments) in a script.

6.1 Loading data in SIS format

Make sure the data file is in the current directory, then load the image using the following code:

```
clear; clc; close all;
load('MRI.mat');  % This file is in the "Image Processing Toolbox".
Data.Images = squeeze(D); % squeeze to remove singleton dimensions
Data.ImageType = 'intensity';
```

We here consider the data as "intensity" data. Consider the Appendix to see the different types of data allowed. The correct metric axes must be generated from the information about the image. In this case, the voxel distance is 2.3 mm in all three spatial dimensions, so the first axis is defined as:

```
Data.Axes(1).Axis = (0:127)*2.3;
Data.Axes(1).Label = 'Vertical axis';
```

```
    Data.Axes(1).Symbol = 'y';
    Data.Axes(1).Unit = 'mm';
```

The second axis as:

```
    Data.Axes(2).Axis = (0:127)*2.3;
    Data.Axes(2).Label = 'Horizontal axis';
    Data.Axes(2).Symbol = 'x';
    Data.Axes(2).Unit = 'mm';
```

And the third:

```
    Data.Axes(3).Axis = (0:26)*2.3;
    Data.Axes(3).Label = 'Depth axis';
    Data.Axes(3).Symbol = 'z';
    Data.Axes(3).Unit = 'mm';
```

Some information about the image is required

```
    Data.ImagesLabel = 'FA map';
    Data.ImagesSymbol = 'FA';
    Data.ImagesUnit = '-';
```

The contents of three lines above is just some text that does not give so much meaning without the original data being more formally described (so do not think more about this, in this tutorial). The last fields that we need are:

```
    Data.Date = now;
    Data.Object = 'Brain';
    Data.Operator = 'Your name';
    Data.Where = 'in my computer';
    Data.ScannerType = 'Test';
    Data.ScannerId = 'Matlab';
    Data.Settings = []; % just empty, but can contain specific info about scanner
    Data.SisVersion = 2;
    save('Data');
```

The data structure is saved, so you can always return to this point.

6.2 Zooming on data

A part of the image can be extracted by use of `sis_zoom`. First, use `sis_view` to visualize the 3D image. The different 'slices' represent the depth axis. Choose the image (slice) that you want to visualize.

```
    sis_view(Data) % choose an image
```

Using `sis_zoom` you can now extract this image.

```
    idx = 1; % This will show the first image; maybe choose another number
    Data_zoom = sis_zoom( Data, [-inf -inf idx], [inf inf idx], 'mmi');
    sis_view(Data_zoom);
```

Here, `[-inf -inf idx]` is the starting point in 3D (thus three coordinates) and `[inf inf idx]` is the corresponding ending point in 3D. `'mmi'` indicates that the first two numbers should be interpreted as metric values, while the last is simple index – this should be the image number you chose above!

Note that both *image* and *axes* are zoomed on.

6.3 Exploring data

A way to visualize 3D data is to use `sis_explore` to scroll through the data (note that you cannot execute MATLAB commands while the `sis_explore` window is open):

```
sis_explore(Data)
```

Another way to use `sis_zoom` is to find the slice using metric values. The metric value corresponding to the slice shown by `sis_explore` is shown above the image. Furthermore, `sis_zoom` can zoom in all dimensions in one single call. Try cropping the horizontal and vertical axes to remove some of the irrelevant parts of the image, while choosing only one image in the depth axis in one command:

```
Data_zoom2 = sis_zoom( Data, [y1 x1 z1], [y2 x2 z1], 'mmm');
sis_view(Data_zoom2);
```

A subfunction of `sis_explore.m` uses `suptitle.m`. This function is build-in to MATLAB if you have the right toolbox. If not, download from: home.healthtech.dtu.dk/jw/jwpublic/matlab_all/contrib.

6.4 Interpolating data

The function `sis_interp3` can be used to improve the appearance of an image. First, a small 3D image is extracted using `sis_zoom` in order to make the interpolation run faster.

```
Data_zoom3 = sis_zoom(Data,[64.4 69 15.9], [142.6 140.3 89.7], 'mmm');
sis_explore(Data_zoom3);
```

The function `sis_interp3` is then given a factor that either enlarges or diminishes the size of the images in the struct. Type `help sis_interp3` in MATLAB for more details about this function and how to call it.

```
Data_interp = sis_interp3(Data_zoom3,[4 4 4]);
sis_explore(Data_interp);
```

The appearance of the image has now been improved to appear more visually pleasant (however, the image does not contain more information).

6.5 Reordering and rotating data

The function `sis_reorder` reorders the order of arguments in the data struct. Here the second axis (horizontal axis) becomes the depth axis, allowing the user to scroll through the data in another direction.

```
Data_reordered = sis_reorder(Data,[1 3 2]);
sis_explore(Data_reordered);
```

Here, it makes good sense to rotate the data, so we can view it as we are used to. For this purpose, the function `sis_rot90` is used:

```
Data_rotated = sis_rot90(Data_reordered);
sis_explore(Data_rotated);
```

The user should note that this function rotates the data counterclockwise along the third axis. To rotate the data a few degrees, the function `sis_rotate` is used:

```
Data_rotated2 = sis_rotate(Data_rotated,[0 0 10]);
sis_explore(Data_rotated2);
```

Here the data can be rotated in all three dimensions at once. Note that this function should only be used for minor adjustment. To rotate 90,180 or 270 degrees, the `sis_rot90` function should be used. See help `sis_rotate` for more information.

6.6 Changing the zero point of the axes

The axes of the image can be changed, so the zero point is at the center of the image. This can be done by changing each of the axes so the zero point is at the center. The first axis can be called by `Data.Axes(1).Axis`, the second by `Data.Axes(2).Axis`, and so on. Redefine each of the axes and plot the image using `sis_explore`, to see if the axes changed as intended.

6.7 Collapsing data

The data can also be collapsed using `sis_collapse`. In this course this will only be used on the 3D CT data, in order to compare this to the 2D X-ray data. The reader should therefore note that collapsing an MR image as done below might not give much relevant information; it only serves to illustrate the use of this function.

```
Data_collapsed = sis_collapse(Data2,3,'sum');
sis_view(Data_collapsed);
```

The above code returns the sum over the specified axis (here depth) by adding the data along the specified axis. The function can also use other methods for collapsing the image, type `help sis_collapse` for more information.

7 Using MATLAB VolumeViewer

Try making a 3D Gaussian function by executing:

```
Specs.Function = 'gauss';
Specs.alfa = 3;
Specs.Width40dB = 30;
Data = sis_function( [20 20 16], Specs);
sis_view(Data);
```

This shows the 3D images as 16 slices. Alternatively, one can use

```
figure; sis_explore(Data);
```

where the wheel on the mouse can be used to go through the same 16 slices. Finally, you can use a new MATLAB function to even more interactively exploring the data:

```
volumeViewer(Data.Images);
```

Press the “Slice Planes” bottom in the menu bar at the top of the figure. Try operating the three sliders at the left part of the figure window. Neat, right?

8 Representing photographs in SIS

Photographs can be represented in MATLAB and in SIS two ways: as an RGB (red, green, blue) image and as an indexed image. But only an indexed image can be plotted with SIS (due to memory considerations). In an RGB image, there will be three images, each of them specifies the intensity of the red, green and blue gun on the CRT monitor in order to show all possible colors. In an indexed image, the image values, which are integers, are indices into a colormap, that contains a limited number of colors specified by RGB triplets.

Start out with a small RGB-image:

```
close all; clear all;
Data.Images(:,:,1) = zeros(50,50);
Data.Images(:,:,2) = ones(50,50);
Data.Images(:,:,3) = ones(50,50);
Data.Images(1,1,1) = 1;  Data.Images(1,1,2) = 0;  Data.Images(1,1,3) = 0;

Data.ImageType = 'rgb';
```

This will be a cyan colored image since all (except one) of the red pixels are set to zero, while the remaining (except one) green pixels are set to one. Likewise for the blue pixels. There will be a red pixel at the upper left hand corner. To verify, you can plot directly in MATLAB

```
imshow(Data.Images)
```

But here, there is no axis on. The metric axis would have to be added manually. So let's continue with SIS. The remaining data in this example is:

```
Data.Axes(1).Axis = 0:49;
Data.Axes(1).Label = 'Vertical axis';
Data.Axes(1).Symbol = 'y';
Data.Axes(1).Unit = 'mm';
Data.Axes(2).Axis = 0:49;
Data.Axes(2).Label = 'Horizontal axis';
Data.Axes(2).Symbol = 'x';
Data.Axes(2).Unit = 'mm';
Data.Axes(3).Axis = 1:3;
Data.Axes(3).Label = 'rgb';
Data.Axes(3).Symbol = '-';
Data.Axes(3).Unit = '-';
Data.ImagesLabel = 'Magnitude';
Data.ImagesSymbol = '|g_r(t)|';
Data.ImagesUnit = 'V';
Data.Date = now; % see help now for information on format
Data.Object = 'colour image';
Data.Operator = 'Your name';
Data.Where = 'in my computer';
Data.ScannerType = 'Simulation';
Data.ScannerId = 'Matlab';
Data.Settings = [];
Data.SisVersion = 2;
```

Since an RBG image cannot be plotted in SIS, you will have to convert it to an indexed image:

```
Data_indexed = sis_rgb_to_icm( Data, 10);
```

where we specify that we want 10 unique colors (which are eight too many in this particular case). Please inspect the content of `Data` and `Data_indexed` (just type `Data` to see the content of `Data`). Which colors are represented in the `Data_indexed.ColorMaps` (which has three columns, one for each of the three colors: red, green and blue). What is the value of the first pixel and all the other pixels? Why can we have a value of zero, when the image values are indices into `Data_indexed.ColorMaps` and MATLAB starts indexing at 1? (answer: because `Data_indexed.Images` are of type `uint8` and the function `imshow` knows this, so it can add one before indexing). Finally, consider the difference between `Data.ImageType` and `Data_indexed.ImageType`!

Now plot:

```
sis_view( Data_indexed);
```

where `sis_view` automatically plots this with the colormap stored in `Data_indexed.ColorMaps`, *i.e.,* as a photograph.

8.1 Flow in programs with rgb data

When working with rgb images (photographs), then the flow in the program should be in this order:

```
sis_import

sis_rotate  (Comment: this function only works on rgb and intensity data)

sis_rgb_to_icm

sis_zoom (Comment: this function only works on indexed and intensity data)

sis_view (Comment: this function only works on indexed and intensity data)
```

9 Problems?

SIS is an experimental research toolbox and undergoes constant adjustments. This, and because MATLAB also is updated with new versions twice a year, means that there can occur errors and/or wrong functionality. However, if a problem occurs, please check the input data with `sis_consistency_check` and also inspect each element of the structure carefully. Another useful function in this regard is `sis_extract_axis_info`. If the problem persists, contact the author of this document.

9.1 Various points

The axis a SIS data structure must always be increasing (*i.e.,* higher indices correspond to higher axis values). If you want to mirror an image, you must operate directly on the image, as is *e.g.,* done with `sis_flipdim`.

If something will not work, always run the data through `sis_consistency_check`.

10 Appendix

10.1 The three data types

In this toolbox, there are three types of data, specified in Data.ImageType

10.1.1 `'intensity'`

Here data is just numbers that specify a values, such as e.g., Hounsfields units. “Intensity” here just refer to the fact that data is shown with a certain intensity on the screen. This is normally used for all the images produced by scanners. The intensity image is normally shown in gray scale, but an color scale can be used.

10.1.2 `'rgb'`

This is normally used for photographs. Any image shown on a computer screen is made up (of a large number of) pixels, where each pixel is illuminated by a certain about of red, green and blue color. If the pixel is in gray scale, these three values are identical, e.g., `[0 0 0]` for black, `[0.5 0.5 0.5]` for medium gray and `[1 1 1]` for white. If the color is pure red, the value is `[1 0 0]`. A photograph of e.g., 100 by 100 pixels would then need three "layers", so that the array is 100 by 100 by 3.

10.1.3 `'indexed'`

If there is a just a few colors in the photograph, e.g., 10, then there is a more effective way of storing the data in the rgb image above. In the indexed case there is just one layer and each pixel is represented by an index value. This index value is in turn used to find the color in a colormap, which in this case will be 10 by 3, where the "3" specifies the rgb colors.

10.2 Overview of data types and sizes

This appendix shows the different formats of `Data.Images` that are allowed (size of image is just an example). The type of image is noted in `Data.ImageType`. Only `intensity` and `indexed` images can be visualized with `sis_view`, but you can work with all three formats. Notice, that for indexed images, if there are *e.g.,* 5 images in `Data.Images`, there are also five (equal size) colormaps in `Data.ColorMaps`; one for each image.

10.2.1 Intensity (2D)

```
Data.Images = ones( 5, 5);
Data.ImageType = 'intensity';
(can be plotted directly with sis_view)
```

10.2.2 Intensity (3D)

```
Data.Images = ones( 5, 5, 5);
Data.ImageType = 'intensity';
(can be plotted directly with sis_view)
```

10.2.3 rgb image (2D)

```
Data.Images = ones( 5, 5, 3);
Data.ImageType = 'rgb';
(can be plotted with sis_view, if Data.ImageType is changed to 'intensity')
```

10.2.4 rgb image (3D)

```
Data.Images = ones( 5, 5, 5, 3);
Data.ImageType = 'indexed';
(can not be plotted with sis_view)
```

10.2.5 Indexed image (2D)

```
Data.Images = ones( 5, 5);
Data.ImageType = 'indexed';
Data.ColorMaps = ones(64, 3)
(can be plotted direclty with sis_view)
```

10.2.6 Indexed image (3D)

```
Data.Images = ones( 5, 5, 5);
Data.ImageType = 'indexed';
Data.ColorMaps = ones( 64, 3, 5);
(can be plotted direclty with sis_view)
```

11 Hints

11.1 Default font size

The change the default font size of plots to e.g. 22:

```
set(0,'DefaultAxesFontSize', 22);
```

To change the default font name of plots to *e.g., Times New Roman*:

```
set(0,'DefaultAxesFontName', 'TimesNewRoman');
```

11.2 Printing to file for your report

You can save a figure as a bit mapped image and/or copying it as a bit mapped image to you report. This is *not* an optimal choice, however, since bit mapped images are not scalable. This is not necessarily a problem with a medical image, but it is withe accompanying text and lines, etc. Thus, to ensure a proper appearance of the report, *vector graphics* - such as eps - should be used.

You can print a figure to an eps file by[1]

```
plot(1:100);  xlabel('This is LaTeX code: x_2^3');  ylabel('Vertical');
print( '-depsc', 'C:\Users\jw\myfigure.eps');
```

You can subsequently investigate `myfigure.eps` with Adobe Acrobat by loading this file (entire filename must be given, as Acrobat normally only shows pdf files). However, the main purpose of making `myfigure.eps` is to include the file in your report (*e.g.,* referencing to the file from within LaTeX).

If you figure should appear with a different size (or proportions) than default, then use

```
set_figure_size
```

which both can be used to inquire the figure size and set it (two different calls, please see `help set_figure_size`).

Use can also used `printfig`, which saves *all* open figures to eps files, but spaces are not allowed in file and directory names

```
printfig( 'FULL DIRECTORY PATH ENDING WITH \', 'file name' );
```

Other ways of saving a figure to disk for use in the report exist, and some may be better suited for your needs.

11.3 Debugging

The best way to work with MATLAB is to work one line at a time. After executing a line in the command window, check two things:

1. Maybe different format under UNIX and/or IOS.

1. The size of the output. Is it as expected? Example: `size(Data.Images)`
2. The values of the output. Are they as expected? `arrayinf(Data.Images)`

This takes some time in the beginning, but normally saves you a lot of time and frustrations in the end.