SBFSEM-tools Documentation

Sara Patterson, Neitz Lab

Contents

1 Introduction

SBFSEM-tools is a Matlab toolbox developed for serial EM data and connectomics in the Neitz Lab at University of Washington. While SBFSEM-tools was built around the Viking annotation software, many aspects are quite general and could apply easily to other programs and imaging methods.
SBFSEM-tools imports annotation data through Viking's OData service and parses the results into Matlab data types. This happens behind the scenes so the average user can work with familiar objects (neuron, synapse, etc).
Other features include:

- Single neuron analysis: dendritic field area, dendrite diameter, soma size, stratification, synapse distribution
- Group analysis: density recovery profile, nearest neighbor, synapse statistics
- 3D volume rendering of polygon annotations and free-form traces over a stack of EM images. 2D projections of dendritic fields.
- Generate surfaces from IPL boundary markers, compensate for Z-axis misalignments.
- Misc UIs for visualizing EM images and annotations

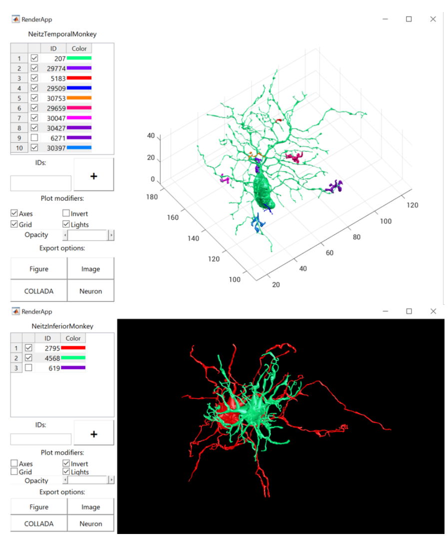
2 Install

SBFSEM-tools is available on Github: sbfsem-tools. You can either clone the repository or download it as a ZIP file. Make sure sbfsem-tools is on your MATLAB path either by adding the following to your startup file, or running it from the cmd line. See Section 9.1 and `Tutorial.m` in the main sbfsem-tools folder for more info on setup.

3 RenderApp

RenderApp includes most of the methods relating to 3D rendering but requires minimal interaction with the command line. To get started type:

```
RenderApp();
```



The user interface was roughly based on VikingView. The left panel is the user interface panel:

- The volume name is displayed at the top of the left panel.
- Each neuron will appear as a line in the table at the top of the left panel. This table offers two controls:
  - The checkboxes toggle visibility of the render. This way you can temporarily hide renders without having to reload the neuron later.

- Click on the color bar to open a UI for changing the render colors.

- Input neuron IDs to the edit box under `IDs`, and press the + button to add them. Add multiple neurons by separating their ID numbers with commas. The loading status of the render will be printed to the command line.

- The plot modifiers include:
  - **Axes** - Show/hide the axes.
  - **Grid** - Show/hide the grid.
  - **Invert** - Switch the background colors from white to black
  - **Lights** - Turn the figure lighting on and off. When off, the render will look like a 2D projection.
  - **Opacity** - Control the render transparency.

- The export options:
  - **Figure** - Open the renders in a new Matlab figure.
  - **COLLADA** - Export the renders as a .dae file for Blender.
  - **Image** - Save the render as an image (either .png, .jpeg or .tiff).
  - **Neuron** - Send the Neuron objects to the base workspace to interact with through the command line.

The right panel is the render display figure. By clicking anywhere on the render figure, you enable the following keyboard controls:

KEY	**FUNCTION**
`h`	Opens help box
`c`	Copies Viking location of last mouse click.
`m`	Return to original axis scaling
Rotate	
←	Decrease azimuth
→	Increase azimuth
↓	Decrease elevation
↑	Increase elevation
Zoom	
`z`	Zoom
`Z`	Hit once to change the zoom direction
Pan	
`a`	Decrease the x-axis
`d`	Increase the x-axis
`q`	Decrease the y-axis
`e`	Increase the y-axis
`s`	Decrease the z-axis
`w`	Increase the z-axis

The biggest difference is that RenderApp does NOT include synapses. Personally, I always have child structures turned off in VikingView - I'll get around to adding them eventually, but if this is a priority for you, let me know.

4 Neuron

Most work revolves around the Neuron class, which is a basic representation of a neuron (called a 'Structure' in Viking). A Neuron is created with two inputs: the Structure ID and the volume name. The Neuron class imports the data from Viking's OData service (so you will need internet access).

```
% Cell 6800 from NeitzTemporalMonkey
c6800 = Neuron(6800, 't');
% Cell 2795 from NeitzInferiorMonkey
c2795 = Neuron(2795, 'NeitzInferiorMonkey');
% Same cell but using the abbreviated volume name
c2795 = Neuron(2795, 'i');

% Import a neuron with synapses
c6800 = Neuron(6800, 't', true);
% Add the synapses to an existing neuron
c2795.getSynapses();
```

The full volume names are: 'NeitzTemporalMonkey', 'NeitzInferiorMonkey' and 'MarcRC1'. These can be abbreviated to 't', 'i' and 'r', respectively.

4.1 Neuron Properties

Neuron has the following publicly accessible properties:

1. `viking` - struct - cell info from Viking
2. `nodes` - table - all annotations
3. `edges` - table - links between annotations
4. `volumeScale` - vector - units are nm/pix for X and Y, nm/section for Z.
5. `synapses` - table - all child structures
6. `geometries` - table - closed curve geometries
7. `analysis` - containers.Map - keys for each NeuronAnalysis
8. `model` - a 3D model of the neuron
9. `lastModified` - datestr - last update of neuron from OData

4.2 Neuron Methods

The Neuron class has the following publicly available methods. Most are convenience functions to simplify data access.
Type `help Neuron\methodname` for more information on each method. Most are also addressed later in the documentation.

1. **update**
   Description: Updates the underlying data from OData. Useful when working with a neuron you are currently annotating.
   Syntax: `obj.update();`

2. **getSynapses**
   Description: Import the synapses ('child structures') from OData.
   Syntax: `obj.getSynapses`

3. **getGeometries**
   Description: Retrieve closed curve annotation control points from OData - helpful when you need to update frequently but don't want to reload the entire cell.
   Syntax: `obj.getGeometries();`

4. **build**
   Description: Create a 3D model (saved to obj.model). Available render types are cylinder (default), closedcurve and disc.
   Syntax: `obj.build(renderType);`

5. **graph**
   Description: Convert the neuron into an undirected or directed graph. For more information on what this enables, see MATLAB's graph class documentation.
   Syntax: `G = obj.graph();`

6. **render**
   Description: Render the neuron's 3D model.
   Syntax: `obj.render('ax', axisHandle, 'FaceColor', rgb);`

7. **printSyn**
   Description: Print a summary of the neuron's synapses to the command line.
   Syntax: `obj.printSyn;`

8. **synapseNames**
   Outputs a list of synapse types associated with the neuron
   Syntax: `names = obj.synapseNames;`

9. **getCellNodes**
   The node table contains both synapse and cell body annotations. This method returns only the cell body annotations.
   Syntax: `T = obj.getCellNodes;`

10. **getSynapseNodes**
    Same idea as getCellNodes, returns only the synapse annotations.
    Syntax: `T = obj.getSynapseNodes(onlyUnique);`

11. **getSynapseXYZ**
    Returns the XYZ locations of all annotations associated with a given synapse.
    Syntax: `xyz = getSynapseXYZ('synapseName');`

12. **getCellXYZ**
    Returns the XYZ locations of all cell body annotations
    Syntax: `xyz = obj.getCellXYZ;`

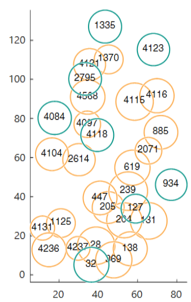
13. **synapseIDs**
    Returns the location IDs for annotations of a given synapse type.
    Syntax: `IDs = obj.synapseIDs('synapseName');`

14. **saveNeuron**
    The speed of the OData import should make saving Neurons unnecessary, however, this is an option just in case.
    Syntax: `obj.save();`

4.3 NeuronGroup

A single data object to hold related Neurons. Inputs can be ID numbers or existing Neurons.

```
h1hc = sbfsem.NeuronGroup([28, 447, 619]);
```



The NeuronGroup class is still a work-in-progress. The current methods include:

somaPlot Plot the mosaic of somas

```
h1hc.somaPlot();
h1hc.somaPlot('addLabel',true); % Label with ID
h1hc.somaPlot('ax',gca); % Add to existing axis
% Two methods for controlling plot color:
h1hc.somaPlot('Color', [0 0.8 0.3]);
h1hc.setPlotColor([0 0.8 0.3]); h1hc.somaPlot;
```

getSomaSizes Return statistics on the soma sizes. If the NeuronGroup contains cells without annotated somas, set validateSizes = true to exclude them from the analysis.

```
% output = NeuronGroup.somaDiameter(validateSizes);
x = h1hc.somaDiameter();
% Catch neurons without annotated somas
x = h1hc.somaDiameter(true);

```

4.4 NeuronAnalysis

The NeuronAnalysis class helps keep population data organized by managing input parameters and results of common analyses. To create a new analysis, subclass NeuronAnalysis and edit the `doAnalysis` and `visualize` methods.
See `Tutorial.m` for information on these existing analysis classes:

- **DendriticFieldHull** - uses convex hull to estimate dendritic field area, includes methods for removing axons prior to analysis.

- **PrimaryDendriteDiameter** - returns the median dendrite diameter at a given distance from the soma.

5 Views

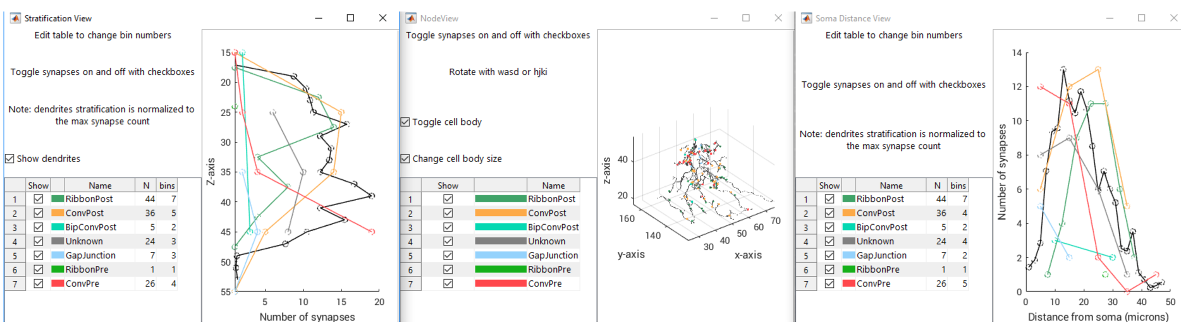
Neurons can be passed to several UIs with the following syntax:

```
% Import a neuron with synapses
c6800 = Neuron(6800, 't', true);
% Pass the neuron to each view
NodeView(c6800);
StratificationView(c6800);
SomaDistanceView(c6800)
```

- **NodeView** - 3D scatter plot of cell and synapse annotations associated with a cell.

- **StratificationView** - Z-axis histogram of dendrites and synapses.

- **SomaDistanceView** - proxmial-distal distribution of dendrites and synapses.



The checkboxes in the synapse table allow toggling the visibility of each synapse type. If the bin numbers in the synapse table are edited, the histogram plots will automatically update.

6 Images

6.1 ImageStackApp

The ImageStackApp was built to scan through a series of images exported using Viking's Export Frames option (although this will work with any folder of images, provided there is some numbering in the file names). Pass a folder name to browse through the images (using left and right arrow keys or buttons). If the filenames contain numbers (Viking's export frames appends a frame number automatically), these will be used to order the images.

```
ImageStackApp('C:\...\foldername');
```

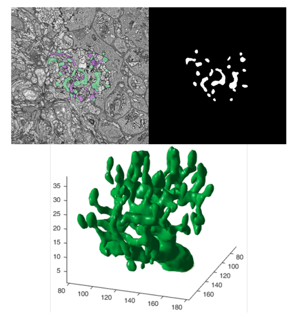
To crop the set of images, use `Process Image` → `Crop` and draw a rectangle on the image. Use `Full size` to return to the original size. Both changes require moving to another image to apply.

The ImageStackApp creates an ImageStack object. To create a GIF, create an instance of ImageStack alone and use the `stack2gif` function.

```
I = sbfsem.images.ImageStack('C:\...\foldername');
I.stack2gif();
```

6.2 Image Segmentation

6.2.1 Color ROIs



Renders can also be generated from a stack of EM images with color annotations. See `algorithms/segmentColorROI.m` for more information. Eventually I will make this more user-friendly. For now, new cell types and their thresholds must be added manually. To the right is an example of an L/M-cone with the ON- and OFF-midgets colored in Photoshop. On the right is the binary matrix output where 1/true (white) indicates a pixel within the color threshold limits.

6.2.2 Outlines

`getImageRGB.m` provides a quick tool for extracting data points from an image, given the outlines were drawn with a distinct color (red, green or blue in Paint works just fine). The first argument is the image, the second is the RGB value used to draw outlines.

```
pts = getImageRGB(im, [0 1 0]);
```

If more than one outline is present, MATLAB's `bwboundaries` can probably separate them into distinct objects.

7 Renders

7.1 Polygon Meshes

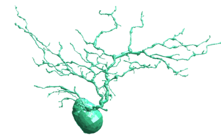
Unlike the volumetric renders, the polygon meshes retain their absolute XYZ locations. This greatly simplifies the process of rendering multiple neurons to a single scene.

```
% Import a neuron and build the model
c1441 = Neuron(1441, 'i');
c1441.build();
% If no axis handle is provided, creates new figure
r1441.render();
% Keep axis handle to direct next render to same figure
ax = gca;
% Import a 2nd neuron and add to render
c1411 = Neuron(1411, 'i');
c1411.build();
```

```
r1411.render('ax', ax, 'FaceColor', [0 0.8 0.3]);
```

The resulting render is passed through a smooth function automatically. You can bypass this or increase the number of smoothing iterations.

```
% No smoothing
c121.setSmoothIter(0);
c121.render();
% Increase the iterations
c121.model.setSmoothIter(2);
c121.render();
```



Surprisingly, increasing the **smoothing iterations** is typically detrimental. The noise in these renders is resistant to a number of methods, including the standard Laplacian and subdivision smooth functions. I'm working on more advanced methods for smoothing out the renders (and on creating them with less noise in the first place).

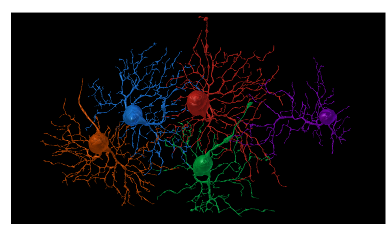
You can also set a **reduction factor** which goes to MATLAB's `reducepatch` function (`help reducepatch` for more info). As of 3Jan2017, the current Cylinder algorithm renders show minimal changes with 0.8, 0.9 reduction factors. To visualize the effect:

```
c121 = Neuron(121, 't');
c121.build();
c121.model.setReduction(0.8);
c121.render('reduce', true);
```

7.1.1 Blender Export

Single neuron To export a single neuron to a COLLADA file, use:

```
c619 = Neuron(619, 'i');
c619.build();
c619.dae();
```



This function will open up a File Explorer window that allows you to select where to save the .dae file.

To export all the neurons in a current figure, use `exportSceneDAE`:

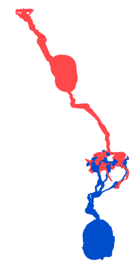
```
% exportSceneDAE(axesHandle, filename, reductionFactor)
% Export the render at max resolution
exportSceneDAE(gca, 'filename.dae');
% Reduce the file size
exportSceneDAE(gca, 'filename.dae', 0.8);
```

These COLLADA files are imported into Blender with `File` → `Import` → `COLLADA (default) (.dae)`. The COLLADA export file is very minimal compared to VikingPlot.

You will have to add a new Material for each cell (the + `New` button in the Materials tab on the right panel). To optimize the final render, I then select `Smooth` for the Shading option in the Tools tab of the left panel. On the `Object Data` tab (directly to the left of the `Materials` tab), check both `Auto Smooth` and `Double Sided`. I move the `Angle` up to 50-60 degrees if the soma has especially sharp angles. Switching the view to Ortho (numpad 5) is helpful too.
Future: apply subdivision smoothing, either automatically in MATLAB [6] or manually in Blender [1].

7.1.2 2D Projection

Rendering the above without any lighting creates a 3D figure with the appearance of a 2D projection.



```
% Turn off lighting for the active window
lighting none;
```

This effect can also be accomplished in Blender with the following steps:

1. Switch from `Blender Render` to `Blender Game` on the top toolbar.

2. On the `Material` tab under `Shading`, check the box for `Shadeless`.

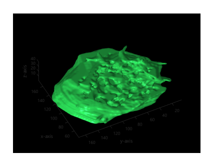
3. On the bottom toolbar, using the button with a circle next to the `Object view` menu, switch from `Material` to `Texture` view.

4. Click anywhere in the viewport, then press `N`. A menu should open to the right. Under `Shading`, check the `Shadeless` box.

7.2 Volumetric Renders

7.2.1 Closed Curve

The closed curve render supports cutouts and multiple annotations per section. These renders lose their absolute position in the volume though. I hope to fix this soon. In the meantime, scenes with both ClosedCurve and Disc renders need to be aligned manually in Blender.

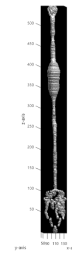


```
c2542 = Neuron(2542, 'i');
% Specify ClosedCurve render
c2542.build('closedcurve');
% Smooth out the render by decreasing the sampling
c2542.build('closedcurve', 'sampling', 0.8);
% Use exportSceneDAE for Closed Curves
exportSceneDAE(gca, 'c2542.dae');
```

7.2.2 Disc Annotations

The same method can be applied to Disc annotations as well for an accurate view of the annotations as discs in 3D.

As is, the result is not as good as the polygon mesh rendering, but could have some advantages in Blender. Right now I'm focusing on the polygon mesh renders for Disc annotations but may revisit this at some point.



```
% Load a rod bipolar cell
c1893 = Neuron(1983, ‘i’);
% Build the 3D model (renders automatically)
c1893.build('disc');

% Adjust the resolution (default = 1)
c1893.build('disc', 'sampling', 0.8);

% Render a subset of sections
c1893.build('disc', 'sections', 1100:1400);
% Like the ClosedCurve renders, COLLADA export requires:
exportSceneDAE(gca, 'filename.dae');
```

7.2.3 Image Traces

Color outlines and transparent overlays can be extracted and rendered using the `volumeRender` function. See 6.2.1 for more information.

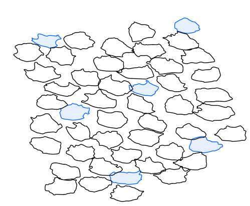
7.3 Outlines

Outlines of single closed curve annotations can also be added to a render using the `outline` option. I use this option for cone mosaics 7.3

```
% Import an L/M-cone:
c5751 = Neuron(5751, 'i');
% Import a neighboring S-cone:
c5752 = Neuron(5752, 'i');
% Build the renders:
c5751.build('outline');
c5752.build('outline');
% Plot both to the same figure
c5751.render('EdgeColor', 'k');
c5752.render('ax', gca,...
'FaceColor', [0, 0.4, 1],...
'FaceAlpha', 0.1,...
'EdgeColor', [0, 0.4, 1]);
```

7.4 VikingPlot

Editing the MATLAB figure generated by VikingPlot can be a good alternative to Blender. A few useful commands:



```
% Make sure the figure is the active window.
ax = gca; % Create a handle to the axis
ax.Children % Returns a list of child structures
% There should be PATCH and LIGHT objects.
% Use numerical indexing to generate handles to the
  objects
% NOTE: The numbering might be different on your figure
```

```
lightObj = ax.Children(1);
renderObj = ax.Children(2);
% A few useful properties to edit for PATCH objects:
renderObj.FaceColor = [0, 0.8, 0.3];
renderObj.FaceAlpha = 0.7;
```

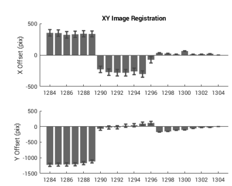
8 Image Registration

8.1 Section Misalignment

The function `xyRegistration.m` calculates the XY offset through a range of Z sections and outputs statistics on the offsets relative to the most sclerad section.

The first output is an Nx3 matrix (columns = Section Number, XOffset, YOffset). A data file tracks these offsets and can be updated using `updateRegistration.m`. The units are Viking's pixel coordinates. The transform is applied when the Neuron class first pulls X,Y coordinates from the OData service. This code is very new and needs quite a few improvements. However, the preliminary results are making a large difference in renders. See the code comments for more info.

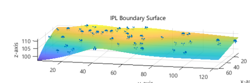
A second method is being used to bridge the gap between sections 915 ('above') and 936 ('below') in NeitzInferiorMonkey. The difference between the last annotation(s) at section 935 and the first annotation at section 915 is subtracted from all annotations above the gap. This is effective for midget RGCs but may need work for larger neurons with multiple branches crossing the gap.




```
[data, S] = xyRegistration('i', [1284 1309], true);
updateRegistration('i', data);
```

8.2 IPL Boundary Markers

The INLBoundary and GCLBoundary objects retrieve all INL-IPL and IPL-GCL boundary markers, respectively. This can be used to create a surface representing the slope of the tissue. In the future, I will add a function for finding the nearest boundary markers to any given Neuron.




```
inl = INLBoundary('i');
% Update marker locations from OData
inl.update();
% Create the surface
inl.doAnalysis();
% Plot the surface
inl.plot();
% Include the data points used to generate the surface
inl.plot(true);
```

Warping the Z-axis to account for the gradually sloping tissue is a big goal. Soon I hope to have code to apply the transform - right now I'm thinking of adapting a commonly used method for warping neurons based on CHaT bands (Github repository) [5].

9 Appendix

9.1 Matlab 101

MATLAB tips are currently scattered throughout the documentation.. I am slowly consolidating them here.
First, using SBFSEM-tools requires adding it to MATLAB's search path. You'll need to fill in the '...' with the full file path where you cloned or downloaded sbfsem-tools.

```
addpath(genpath('C:\...\sbfsem-tools'));
```

The `addpath` command adds all the files in the specified folder to your search path. The `genpath` command adds all the sub-folders their files as well. If you're having trouble with this, there is a user interface (UI). This can be accessed by the `Set path` button on MATLAB's top toolbar, in the `Environment` panel. Make sure to click the `Add with subfolders` button (which is the equivalent of the `genpath` command). You can also open this UI by typing

```
pathtool;
```

into the command line.

For any function or class, the `help` command will print information to the command line. For more information on MATLAB's functions, use `doc` command. To see the raw code, use the `edit` command.
To explore any variable in a UI similar to Excel, use `openvar`. Most of the data types are user-defined classes like Neuron. A list of all the methods for any class can be obtained by `methods(className)`.

```
c127 = Neuron(127, 'i');
% Open the variable in a UI
openvar(c127);
% These are the methods described in Section 3.2
methods('Neuron')
% Return the help for sbfsem.render.Cylinder
help sbfsem.render.Cylinder
% Go to matlab's documentation for tables
doc table
% See the cylinder code
edit sbfsem.render.Cylinder
```

If you're planning on using SBFSEM-tools often, put the `addpath` code in your startup file and it will run each time you open MATLABb.

9.2 Git 101

Git can be downloaded here. Once downloaded, you can open the terminal by right clicking anywhere in the whitespace of a File Explorer window and selecting `Git Bash here`. Do this in the folder you want sbfsem-tools to live in.

```
git clone http://github.com/sarastokes/sbfsem-tools.git
```

There should be a folder now called "sbfsem-tools". At any point, you can update the code to the latest version by right clicking on the folder itself, selecting `Git bash here` and typing in:

```
git pull
```

This saves the step of deleting the existing sbfsem-tools folder and re-downloading it every time an update is available.

References

[1] BornCG Tutorial 8 - Smoothing and SubSurf
https://www.youtube.com/watch?v=gkN1aLaNWxk

[2] ISETBIO documentation format
https://github.com/isetbio/isetbio/wiki/Documentation-&-Formatting

[3] Levy, B., Zhang, H. (2009) Spectral Mesh Processing. SIGGRAPH Asia Course Notes.

[4] Lorensen, W.E. & Cline, H.E. (1987) Marching Cubes: A high resolution 3D surface reconstruction algorithm. *Computer Graphics*, 21(4), 163-169

[5] Sumbul, U., Song, S., McCulloch, K., Becker, M., Lin, B., Sanes, J.R., Masland, R.H. & Seung, H. S. (2014) A genetic and computation approach to structurally classify neuronal types. *Nature Communications*, 5, 3512
Github repository: https://github.com/uygarsumbul/rgc

[6] Zorin, D. & Schroder, P. (2000) Subdivision for modeling and animation. SIGGRAPH Course Notes