*(Words in italics and parentheses are notes to the teacher.)*

Words in red are notes to the writer about things that need fixing or further exploration.

Video 1: Optimization

* MDO: what it is, why it’s useful
* Failed cases: what they are, why they happen, how they’re handled
* Postprocessing optimized data
* Choosing initial guesses for the optimizer and solver
* Differences between gradient-free and gradient-based optimization
* How/when to use equality constraints

Video 2: Introduction to OpenMDAO

* Motivation behind using OpenMDAO
* What OpenMDAO is useful for and can handle
* How OpenMDAO thinks (hierarchical modularized models) (show lots of pretty pictures with lots of explanation to help them grasp the overall structure, this is very important.)
* OpenMDAO has both solvers and drivers. Briefly explain what these do.
* Why OpenMDAO needs derivatives
* Installing OpenMDAO
* General layout of an OpenMDAO file

Video 3: Running with OpenMDAO

* Importing OpenMDAO classes
* Importing components from pre-existing files
* Running a sample model
* Understanding the sample model output
* Accessing and printing out values from the model
* **Assignment: Run the paraboloid model and print out each out the variables**

Video 4: Explicit Components in OpenMDAO *(For this video, have some slides at the beginning illustrating what the component does and what the different methods do. Then type code out on the screen while illustrating exactly what each line does. The paraboloid example should work well for this video.)*

* Explicit components are components where the outputs are defined as explicit functions of the inputs.
* Two necessary methods: setup and compute *(Show on slide what each method does and explain it a little.)*
* Setup is where the inputs and outputs are declared, along with their derivatives
  + Show how to select and add inputs and outputs
  + Show how to add defaults, descriptions, and units to the inputs and outputs
  + Show how to select and declare derivatives, including derivative type. For now we will finite difference all our derivatives.
* Compute is where the actual explicit equation is defined
  + Show how to make the method
  + Show what parameters to pass
* *(Run the model so they can see the output. In order to do this there will need to be a problem defined, so just copy paste the problem code on the screen and quickly explain that it will be demonstrated later how that code was generated.)*
* **Assignment: Build a simple explicit component using the equation . It is only necessary to use the setup and compute methods, just finite difference the derivatives. *(Or use some other simple explicit component that has already been written.)***

Video 5: Implicit Components in OpenMDAO *(For this video, have some slides at the beginning illustrating what the component does and what the different methods do. Then type code out on the screen while illustrating exactly what each line does. The quadratic example from the docs should work well for this video.)*

* Implicit components are components where the outputs are not explicit functions of the inputs.
* These components need to be used if the desired outputs cannot be written as explicit functions of the inputs, but they can also be used at other times too. These times will be addressed later in the course.
* Two necessary methods: setup and apply\_nonlinear *(Show on slide what each method does and explain it a little.)*
* Setup is where the inputs and outputs are declared, along with their derivatives.
  + Explain how to determine which variables are inputs and which are outputs.
  + Show how to add inputs and outputs.
  + Show how to add defaults, descriptions, and units.
* Apply\_nonlinear is where the implicit equation(s) is(are) defined.
* Instead of writing explicit equations as was done in the explicit component, here each equation is defined by a residual. There is a residual for each output, but even though the name of the residual follows the name of the output it corresponds to, the actual value is will be zero, and the output need not appear in its own residual.
* *(Run the model so they can see the output. In order to do this there will need to be a problem defined, so just copy paste the problem code on the screen and quickly explain that it will be demonstrated later how that code was generated.)*
* **Assignment: Build a simple implicit component using the equation . It is only necessary to use the setup and apply\_nonlinear methods, just finite difference the derivatives. *(Or use some other simple explicit component that has already been written.)***

Video 6: Groups in OpenMDAO *(For this video, have some slides at the beginning illustrating what the component does and what the different methods do. Then type code out on the screen while illustrating exactly what each line does. The Sellar example from the docs should work well for this video. Don’t go though and define the components for the problem, just present a slide with the equations and explain that the components have already been written according to the methods previously discussed, then focus on using the group to tie them together.)*

* The basic group operations happen in the setup method.
* Show how groups fit into the OpenMDAO structure.
* Show how to add an IndepVarComp and explain what it does.
* Show how to add components.
* Show how to add subgroups.
* OpenMDAO intrinsically converts units, as long as units, as long as units are specified for each input and output.
* Show how to connect inputs and outputs via connection.
* Show how to add a solver to a subgroup.
* Show how to add an objective and constraint.
* *(Run the model so they can see the output. In order to do this there will need to be a problem defined, so just copy paste the problem code on the screen and quickly explain that it will be demonstrated later how that code was generated.)*
* Another way to write groups is to connect variables via promotion instead of connection. Explain the difference between promotion and connection, and show how to rewrite the problem using promotion.
* *(Run the model again so they can see that the output hasn’t changed.)*
* **Assignment: Find a group example to use. Read the provided group code and the accompanying equation which are in the components. Then modify it so that it uses promotion instead of connection.**

Video 7: Problems in OpenMDAO and review of model hierarchy *(For this video start by saying that today we are going to learn to bring everything together. Then pick one of the models (either a component or group exercise) that we have developed thus far and use it to write a problem for. After that code has been run, use slides to show how each of the models we have developed thus far fits together with its problem, group (if necessary) and solver/driver.)*

* The problem is the container for the entire OpenMDAO model
* Show how to instantiate a problem
* Show how to add components and groups
* Show how to set up a driver
* Show how to add design variables
* Show how to add constraints
* Show how to add objectives
* *(Run the problem so they can see the output.)*
* Show the slide with the general OpeMDAO hierarchy again and remind them of how the overall structure works. Then put up a slide for each of the previous lecture examples showing the hierarchy of how that example would look in a problem. Include any drivers. Talk through each slide and explain why it is broken up the way it is. (The two component slides will look basically the same.)
* **Assignment: Choose one of the previous assignments and write a problem for it, then run it. For extra practice do this for more than one!**

Video 8: Putting it all together *(This video doesn’t teach any new content, it is an extended example on how to use what has been taught so far to put together a simple model.)*

* Find/make a simple model that uses one implicit and one explicit component, and uses both connection and promotion. The Sellar problem would be good in this case, and since the components haven’t been built before in the lecture it should be fine. Just make sure to write one implicit component, even though it is not necessary. However, since vectorization hasn’t been taught yet, make sure to make z1 and z2 separate inputs.
* Show (on a slide) the equations chosen, and how they will be used (ie. what are the inputs and outputs of each component?)
* Show (on a slide) a picture of the model structure for the model to be built.
* Type out the code for each of the two components, making sure to only use the setup and compute/apply\_nonlinear methods, and use fd derivatives. Verbally explain what the code does as it is typed.
* Type out the code for a group to put the two components in, and put a solver in it. Try to use both connection and promotion. Verbally explain what the code does as it is typed.
* Type out the code for the problem, and verbally explain what the code does as it is typed.
* *(Run the model so they can see the output. Make a bit of a fuss over the fact that they now know how to build their own simple model. Get them excited about realizing how much they have learned.)*
* **Assignment: None.**

Video 9: ExecComp, BalanceComp *(For this problem, type the code out on the screen while illustrating what each line does. For the ExecComp the paraboloid example should work well, and for the BalanceComp the quadratic example should work well.) More detail here? There’s a lot more to these components than just one example can cover…*

* ExecComp is a shortcut to an explicit component.
* It doesn’t require all the setup of a regular explicit component, it just needs an equation.
* It assumes outputs on the left and inputs on the right.
* It automatically uses fd for the derivatives. (is this true?)
* Show the code where the paraboloid component is called (from the explicit component video) and show how to modify it to use an ExecComp.
* *(Run the model so they can see the output.)*
* Similarly, BalanceComp is a shortcut to an implicit component.
* It doesn’t require all the setup of a regular implicit component.
* Show the code where the quadratic component is called (from the implicit component video) and show how to modify it to use a BalanceComp.
* *(Run the model so they can see the output.)*
* **Assignment: Rewrite the explicit component from video 4 using an ExecComp. Rewrite the implicit component from video 5 using a BalanceComp.**

Video 10: Adding Vectors in OpenMDAO *(For this problem start out with a few slides talking aout the first couple bullet points. For the coding examples, use the Sellar problem from before and show how to modify it to vectorize everything.)*

* Sometimes it is necessary to have inputs and/or outputs that are in vector form
* Explain what the rows and columns of a Jacobian represent
* Explain that it is easier for OpenMDAO if we specify which values in the Jacobian are nonzero
* Explain how to determine which values in the Jacobian are nonzero
* Show a reminder slide with the equations and modular structure of the Sellar problem
* Show how to modify the Sellar code in the setup method to vectorize the z-variable. Use the code generated in the bringing-it-together example, because this has one explicit and one implicit component, so they can see how it works in both.
* Show how to modify the Sellar code in the compute and apply\_nonlinear methods to vectorize the z-variable.
* *(Run the model so they can see that the output hasn’t changed.)*
* **Assignment: Vectorize the y-variable in the Sellar code and run the model to make sure that the output hasn’t changed.**

Video 11: Derivatives in OpenMDAO *(For the coding in this video use the Sellar problem from before, both the vectorized and un-vectorized versions (to show the different ways of implementing partials in vectorized and un-vectorized situations.))*

* Have slides and discuss the benefits and drawbacks of fd, cs, and analytic derivatives in OpenMDAO. *(There should already be slides on this in “getting\_derivatives\_in\_OpenMDAO.tex” in the OpenMDAO training materials repo. Or if it is preferred, the slides are also in John Jasa’s “getting\_derivatives\_presentation.pptx”.)*
* Derivatives are implemented similarly but slightly differently in Explicit and Implicit Components.
* In order to implement them as cs instead of fd, the only thing that needs changing from before is that the method declared is cs instead of fd. Implementing analytic derivatives is a bit more involved.
* How to declare derivatives was covered in previous videos, but now instead of being declared as fd they will be declared as analytic (the default if no method is specified).
* Show a slide with the partial derivatives of the Sellar problem.
* For Explicit Components, the compute\_partials method is used to define analytic derivatives. It can be used both for vectorized and non-vectorized equations.
* Show how to implement the compute\_partials method for the non-vectorized Explicit Component.
* Show how to implement the compute\_partials method for the vectorized Explicit Component.
* *(Run the model so they can see if the output has changed. Also change the method back to fd as well as to cs, and run the model in all cases so they can observe whether or not the output changes. Afterwards return the method to fd so that the impact of analytic derivatives on the Implicit Component alone can be explored next.)*
* Derivatives in Implicit Components are slightly different, because instead of taking the derivative of outputs with respect to inputs, we take the derivative of residuals with respect to both outputs **and** inputs.
* The mechanics of declaring and providing the analytic derivatives are the same in Explicit and Implicit Components, but in Implicit Components the derivatives are provided in the linearize method.
* Show how to implement the linearize method for the non-vectorized Implicit Component.
* Show how to implement the linearize method for the vectorized Implicit Component.
* *(Run the model so they can see if the output has changed. Also change the method back to fd as well as to cs, and run the model in all cases so they can observe whether or not the output changes. Then go back and turn on the analytic derivatives in the Explicit Component as well as here in the Implicit Component, run the model, and show them if the output changes.)*
* Assignment: Pick two of your favorite assignment models we have worked on so far and implement analytic derivatives in them. Do the results change? Why or why not?

Video 12: The initialize method *()*