Progress Report

Pranav Bangarbale - 1/11/22

Recap - Last Progress Report

- Implemented K-means algorithm for intermediate weight averaging
- Generated results of K-means implementation
- Overall results summary
  - Fits of F were similar to original method 2 (otherwise performed well)
  - Fits of the CFFs performed worse than original Method 2 - possibly due to *smaller numbers of networks, lower number of trained epochs, small number of replicas, small number of k-means clusters or a variety of factors*
- Next goal: experimenting with above variables to see if K-means implementation can produce results better than the original Method 2

Purpose of K-Means (Explanation)

- Instead of training as many epochs, simply narrow the number of networks at an intermediate stage and then continue training
- Goal of K-means is to find representative “samples” of each of the 5, 10, 20, n number of networks that represent the different variations of the overall 500, 1000, etc (depending on number of replicas).
- Training is then continued with the representatives instead of every network
- Goal is to save time and also see if the representative networks can each get closer to the true CFF values without having to look at the results of thousands of networks

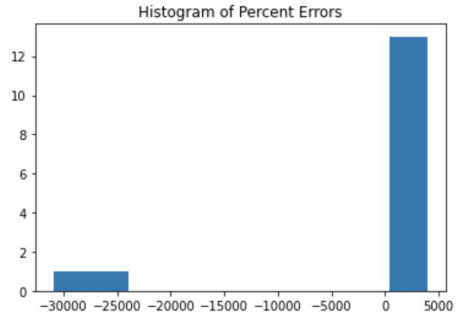
```
# Fit kmeans clustering to group of weights from set replicas
w = np.array(intermediate_models)
nsamples, nx, ny = w.shape
w1 = w.reshape((nsamples,nx*ny))
kmeans = KMeans(n_clusters=5, random_state=0).fit(w1)
cluster_to_models = {}
for model in set_models:
    input = []
    for x in model[2]:
      for y in x:
        input.append(y)
    input = [input]
    closest = kmeans.predict(np.array(input))[0]
    # print(closest)
    # i = np.where(kmeans.cluster_centers_==closest)
    # print(i)
    if closest not in cluster_to_models:
      cluster_to_models[closest] = list()
    cluster_to_models[closest].append(model)
# have each model for set and which cluster each model belongs to
# want to average weights for models of the same cluster and append these cff values to by_set
for c in cluster_to_models:
  group = cluster_to_models[c] # group of models for a given cluster (given by weights)
  avg_weights = np.mean(group, axis=0) # element-wise weight averaging
  globalModel.set_weights(avg_weights)
  globalModel.fit([setI.Kinematics, setI.XnoCFF], setI.sampleY(), # the sample Y will generate Fs
                epochs=100, verbose=0,
                callbacks=[model_checkpoint_callback])
  globalModel.load_weights(chkpt_path) # load back minimum loss epoch
  cffs = uts.cffs_from_globalModel(globalModel, setI.Kinematics, numHL=NUM_HIDDEN_LAYERS) # get cffs from middle model
  by_set.append(cffs)
```

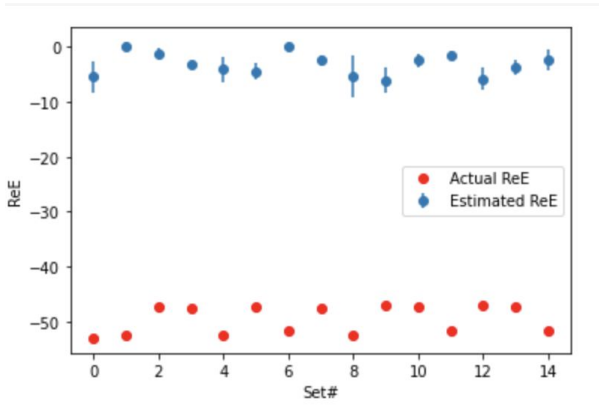
Summary of Progress

- Experimented with different numbers of replicas, clusters, epochs
- Found that # epochs had significantly less influence on final results than replicas and clusters per set
- “Best” results were achieved with using all 15 sets, 1500 replicas, 10 k-means clusters per set, 300 epochs before narrowing and 300 epochs after narrowing
- However, results were not significantly different from those at the smaller scales I presented last time

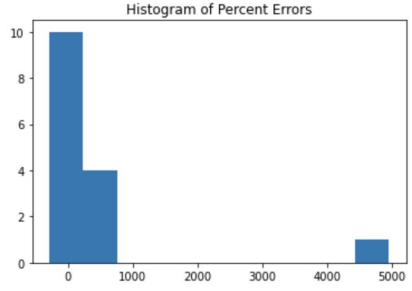
ReE Fit (fit from last time on bottom, new fit on top)

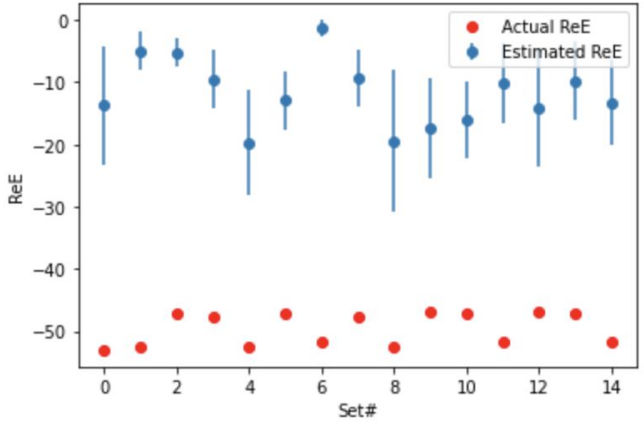
- Mean percent error: 5167.420371605129
  RMSE: 46.58255438657015
  RMSE w yhat=mean: 2.525480507907797
  R-squared: -339.2189951795507





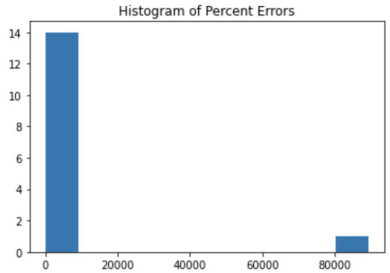
Mean percent error: 477.78184628367933
RMSE: 2.976540931551008
RMSE w yhat=mean: 1.4030345621243816
R-squared: -3.5007716491044656



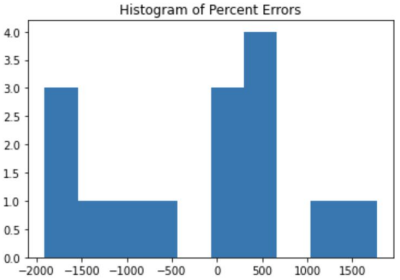


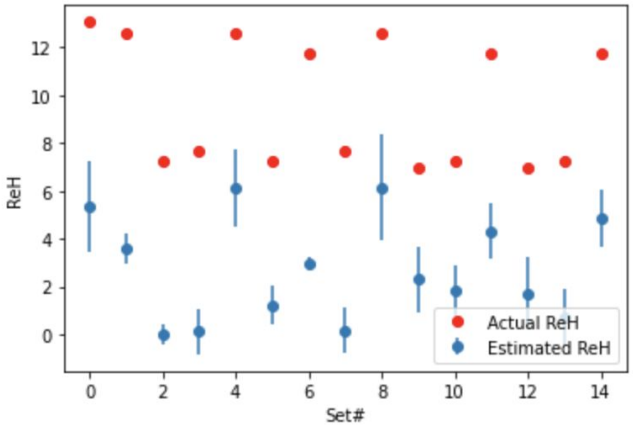
ReH fit

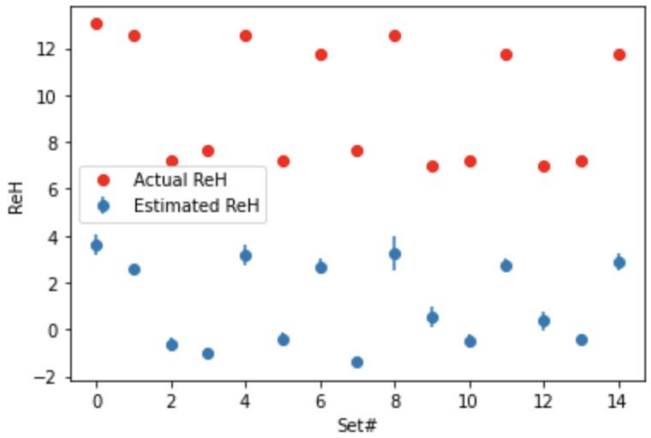
Mean percent error: 6899.937690226596
RMSE: 6.945863981760171
RMSE w yhat=mean: 2.5254628436780107
R-squared: -6.564332008429753



Mean percent error: 886.5477930612573
RMSE: 8.49592975595536
RMSE w yhat=mean: 2.5254628436780107
R-squared: -10.317222635225521

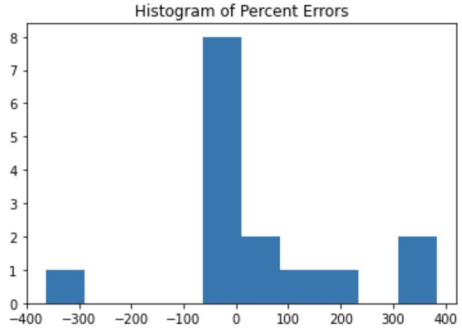


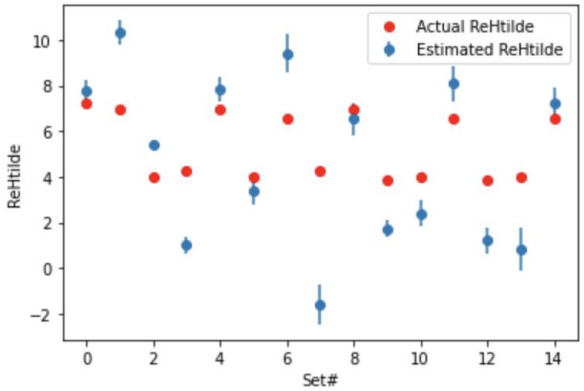




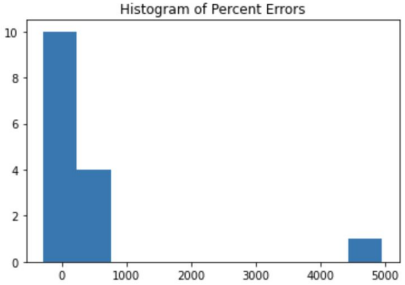
ReHtilde fit

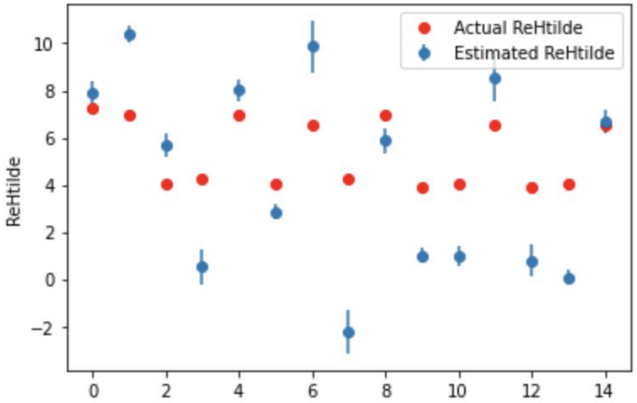
```
Mean percent error:  109.94245150570077
RMSE:  2.5248548827957262
RMSE w yhat=mean:  1.4030345621243816
R-squared:  -2.238441861899217
```





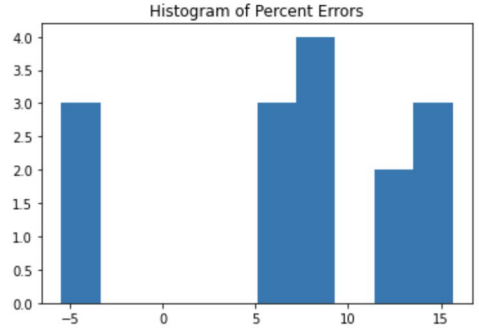

```
Mean percent error:  477.78184628367933
RMSE:  2.976540931551008
RMSE w yhat=mean:  1.4030345621243816
R-squared:  -3.5007716491044656
```



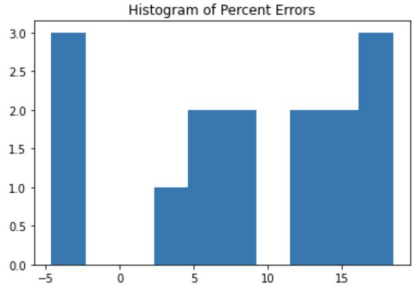


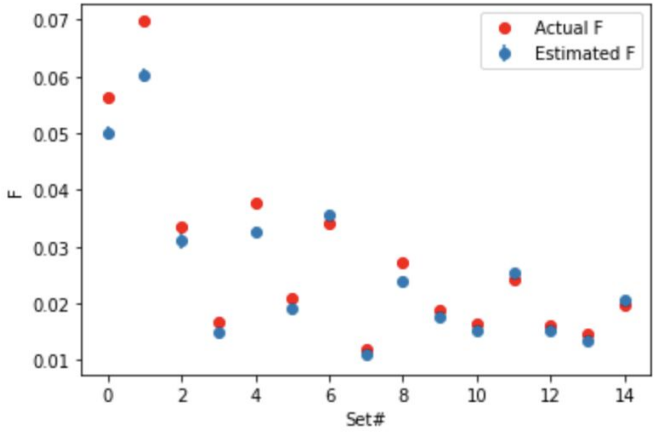
F fit

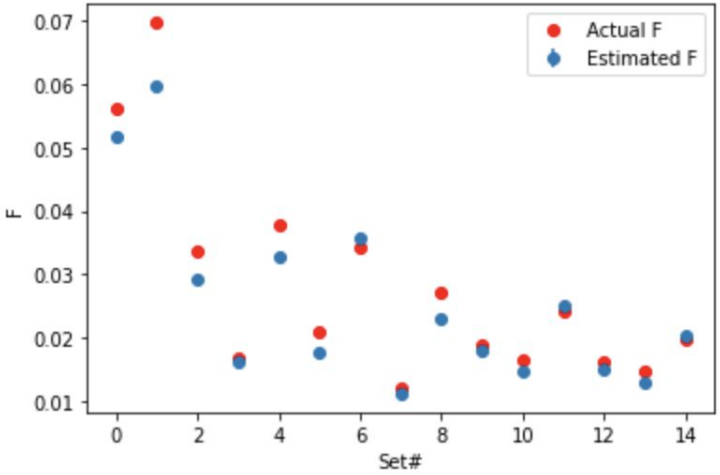
```
Mean percent error:  9.079079331409337
RMSE:  0.003527317163901565
RMSE w yhat=mean:  0.015855649879790133
R-squared:  0.9505096040116555
```




```
Mean percent error:  10.217115631864447
RMSE:  0.0037381169089202248
RMSE w yhat=mean:  0.015855649879790133
R-squared:  0.9444175526949976
```







Possible Reasons for Failure

- (Unlikely) After training for the additional epochs the averaging of weights becomes useless since the averaged weights are merely a starting point
    - After analyzing some of the outputs, I determined this was likely not the case. The outputs from before and after the additional training were different, but similar enough where it would not have significantly affected results
- Simply averaging the weights does not necessarily mean that distributions will be averaged as well (e.g. if an overall distribution with 1000 replicas was narrowed to 20 networks, these 20 with averaged weights might not necessarily reflect the average outputs of those networks, even if they accounted for averaged weights).