WCT estimation

Background

- Create a model to estimate the wall clock time for LF, HF or BB run

Tasks:

- Collect, combine and transform metadata (done)
- Investigate and test a model -> Neural network (done)
- Streamline training and usage of NN (done)
- Add support for number of cores parameter (mostly done, require real data to test)
- Change submit scripts (submit_emod3d.py....) to use pre-trained model
    - Change submit scripts to python3
        - submit_emod3d (done, currently testing that it still works)
        - Created python3 virtual environment for maui and mahuika

    Change all submit script dependencies to be python 2 and 3 compatible (done, currently testing)
- Create script to estimate full run time for a folder of srf/vms
- Train actual model once maui data is available, i.e. after current cybershake run
- (Uncertainty?)
- (Visualisation?)

Documentation

- Also exists as Readme.md in slurm_gm_workflow/estimation/

Notes

- All of the estimation code is written in python3, the only exception is the write_jsons.py script used for metadata collection

Usage

Estimation is done using the functions inside estimate_WC.py, which load the pre-trained neural network and then run the estimation.

A model has to either exist in the default model location ./estimation/models/(LF/HF or BB)/model*xxx.h5 along with pickled StandardScaler (sklearn) scaler*x xx.pickle, or a model directory has to be specified manually

Estimation of wall clock is already included in the slurm script creation for simulation.

For once off estimation the *estimate_LF_WC_single*, *estimate_HF_WC_single* and *estimate_BB_WC_single* functions from estimate_WC.py can be used

```
import estimation.estimate_WC as wc
print(wc.estimate_LF_WC_single(1000, 1000, 100, 2500, 160))
```

The signatures for the estimation functions are:

```
def estimate_LF_WC_single(
        nx: int, ny: int, nz: int, nt: int, n_cores: int,
        model_dir: str = LF_MODEL_DIR, model_prefix: str = MODEL_PREFIX,
        scaler_prefix: str = SCALER_PREFIX):
        pass

def estimate_HF_WC_single(
        fd_count: int, nsub_stoch: float, nt: int, model_dir: str = HF_MODEL_DIR,
        model_prefix: str = MODEL_PREFIX, scaler_prefix: str = SCALER_PREFIX):
        pass

def estimate_BB_WC_single(
        fd_count: int, nt: int, model_dir: str = BB_MODEL_DIR,
        model_prefix: str = MODEL_PREFIX, scaler_prefix: str = SCALER_PREFIX):
        pass
```

These all return the number of core hours, to convert that to the wall clock time format use the *convert_to_wct* function. To convert and add some extra buffer, use the *get_wct* function.

```
def convert_to_wct(core_hours):
    pass

def get_wct(core_hours, overestimate_factor=0.1):
    pass
```

Creating a pre-trained model

Building a pre-trained model consists of a two main steps, **collect and format the data** and then **training the neural network**

Colleting the metadata

1) To create the metadata files for a given Run, use the *write_jsons.py* (with the -sj flag) script, which will create a folder names "jsons" in each fault simulation folder. This jsons folder then contains a "all_sims.json" file, which has all the metadata for the faults realisations runs. E.g.

```
python2 write_jsons.py /nesi/nobackup/nesi00213/RunFolder/Cybershake/v18p6_batched/v18p6_exclude_1k_batch_2/Runs/
-sj
```

2) Combine all the "all_sims.json" files using the *agg_json_data.py* script, which for a given run folder will find all the metadata json files and combine them into a single dataframe. In addition it also tidies up some columns, and attempts to convert them all to floats. If entries are missing then these are either filled with np.nan for numerical columns or None. The resulting dataframe is then saved as a .csv file E.g.

```
python3 agg_json_data.py /nesi/nobackup/nesi00213/RunFolder/Cybershake/v18p6/Runs/ /home/cbs51/code/tmp/output.csv
```

It is worth noting that loading that .csv into a dataframe requires some changes from the default arguments for *pd.from_csv* due to the multi level columns used in the dataframe. Loading the resulting csv:

```
import pandas as pd
input_file = "/home/cbs51/code/slurm_gm_workflow/estimation/" \
             "labelled_data/test/kupe_agg_data.csv"
df = pd.read_csv(input_file, index_col=[0], header=[0, 1])
```

3) Steps 1 and 2 can be repeated for as many run folders as wanted. I would suggest putting all the resulting .csv files into a single directory, for easier loading when training the neural network model.

Training the model

The training of the different models (LF, HF, BB) is done using the *train_model.py* script, which takes a **config file** and the input data (either as directory or single input files). The input data is the saved dataframes from the previous step.

The config file determines the architecture of the neural network, i.e. the columns used for training, target column, number of layers and units, activation function and dropout. It also specifies how to train the model, i.e. number of epochs, loss function and metrics to record.

The code is currently hardcoded to use the "Adams" optimizer, this could be added to the config, but given the problem I don't think there is much reason for ever changing that. And if that's required it's a straightforward addition. The current in use config will be added to git.

The weights of the trained model are then saved in an .h5 file along with a .pickle file, which stores the instance of StandardScaler used for the preprocessing of the data (done as part of the *train_model.py* script).