Lehrstuhl für Energiesysteme
TUM School of Engineering and Design
Technische Universität München

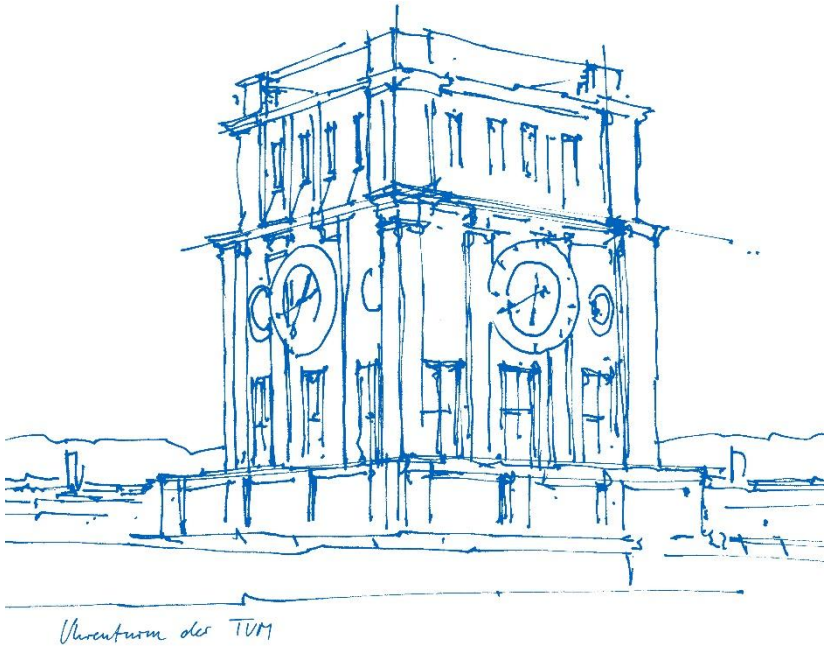
CEA model data Training

Erfan Mashayekh

Technische Universität München

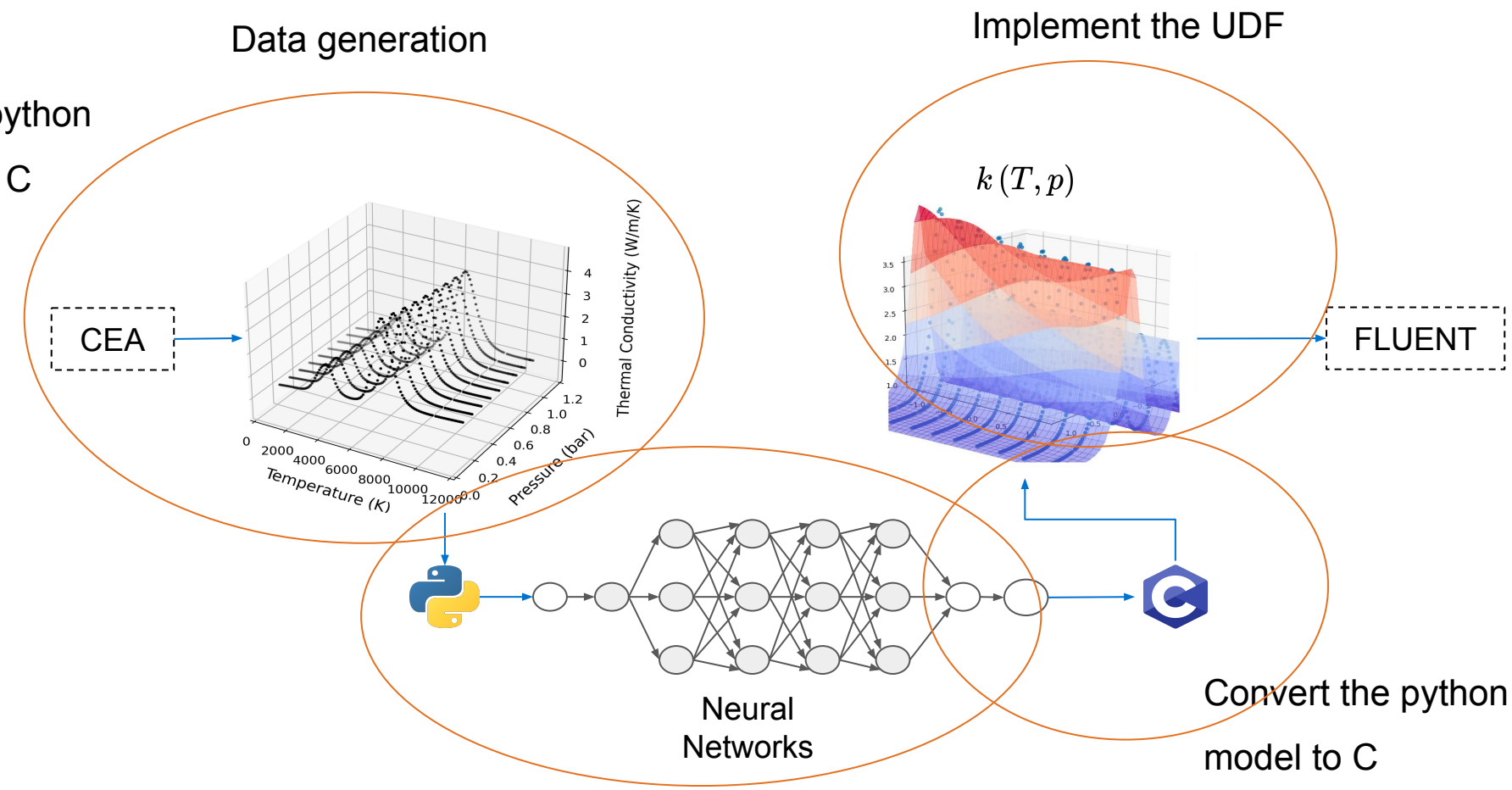
Fakultät für Maschinenwesen

Lehrstuhl für Energiesysteme

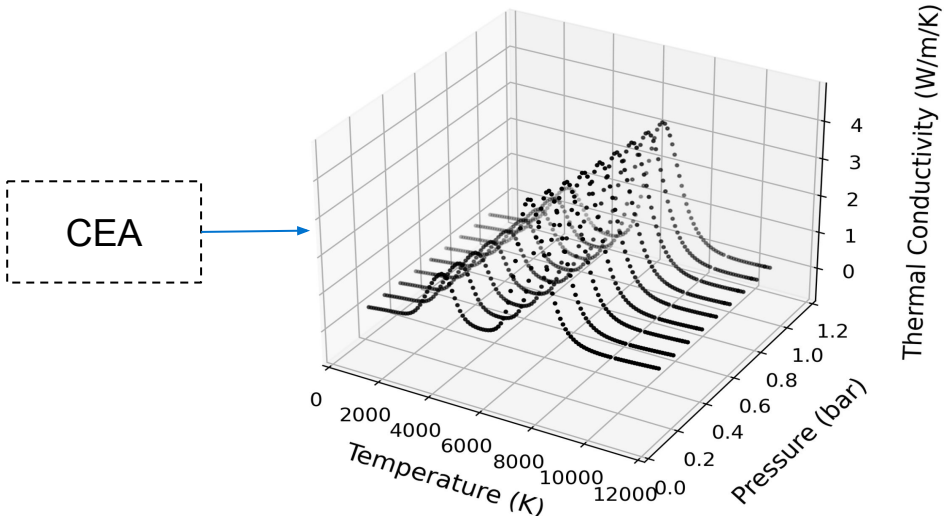


Overall Process

1. Data generation
2. Fit data with a model in python
3. Convert python model to C
4. Implement the UDF



1 Data Generation



1 Data generation

Input File

- Generate a text file and rename it as ***.inp**.
- implement the desired pressure and temperature values.
- Open the input file in CEA.

Limitation: It is only possible to enter 13 temperatures values for each calculations.

1-200mbar.inp

File Edit View

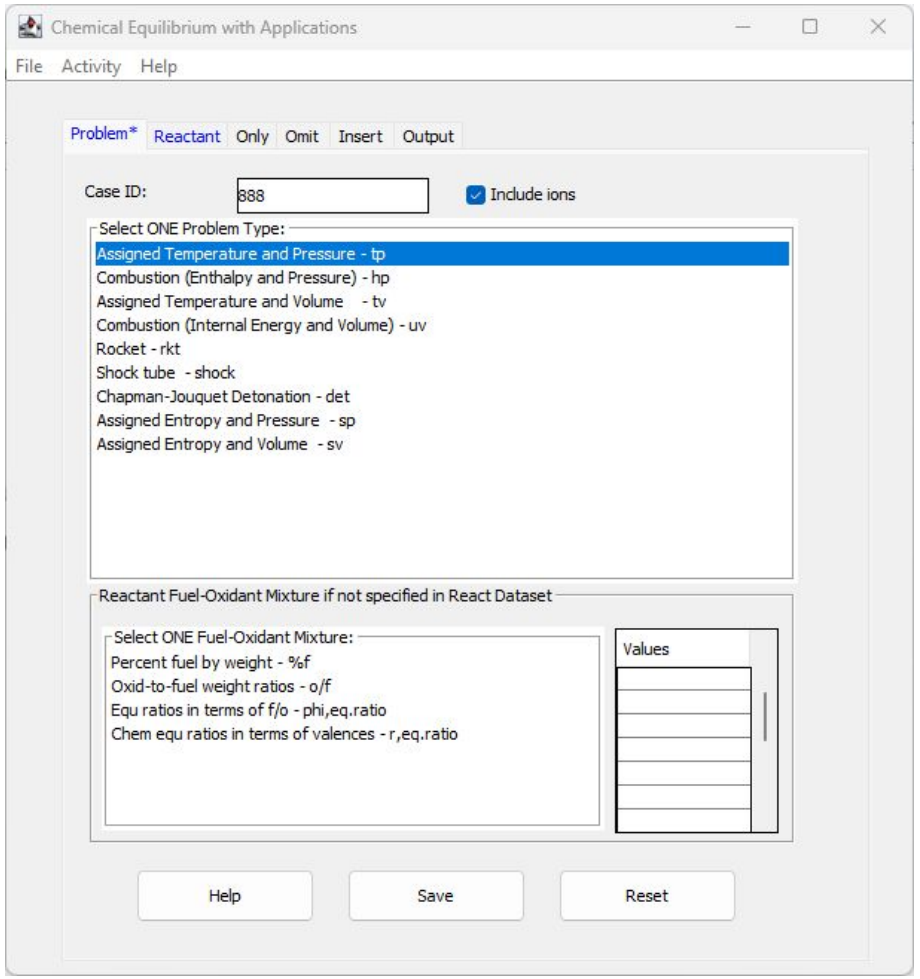
```
problem   case=200 ions
    tp   t,k=200,300,400,500,600,700,800,900,1000,1100,1200,1300,1400,  p,bar=0.2,
react
  fuel=CO2 wt=100
only
  e- C C+ C- CO CO+ CO2 CO2+ C2 C2+ C2- C2O C3 C3O2 C4 C5 O O+ O- O2 O2+
  O2- O3
output  transport
    plot p t rho h u g s mw cp son vis cond pran
end
```

Ln 11, Col 1 | 100% | Windows (CRLF) | UTF-8

1 Data generation

Problem

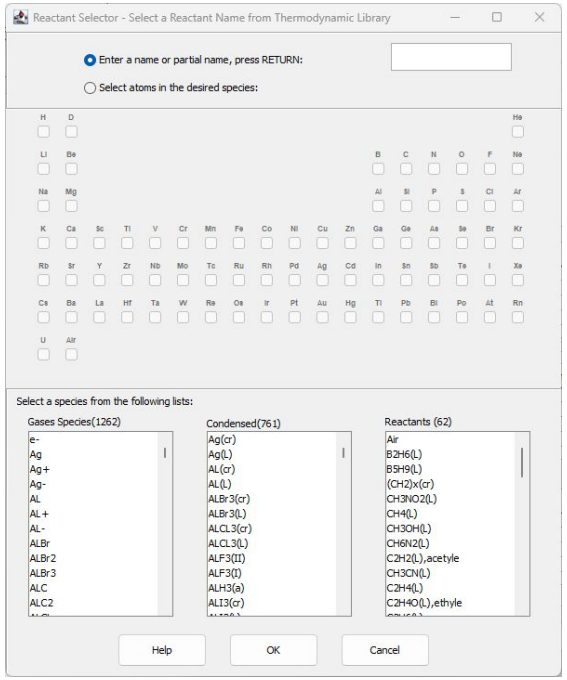
- Select problem type as: Assigned Temperature and Pressure - tp
- Give an arbitrary number to Case ID: 888, or …
- In case of including ions, check the **Include ions** box

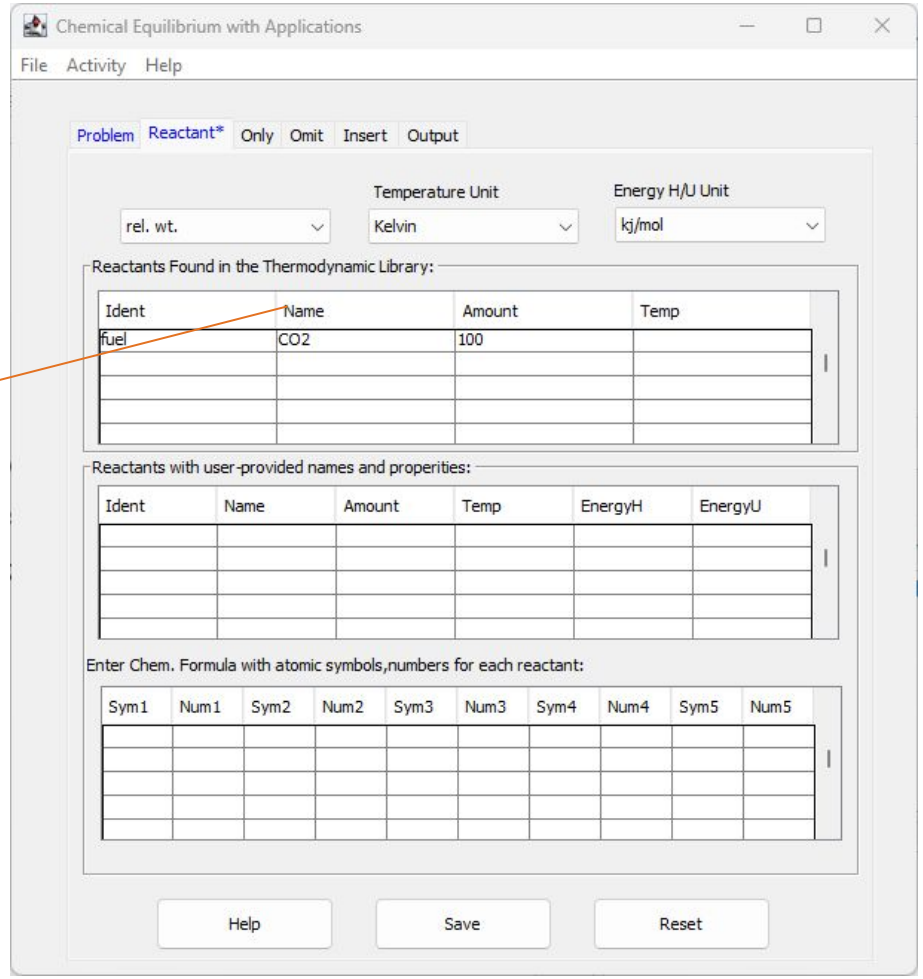


1 Data generation

Reactant

- Set the Input gases/ions as reactants with their mass/molar fractions
- Click on **Name** and the choose the desired gases/ions

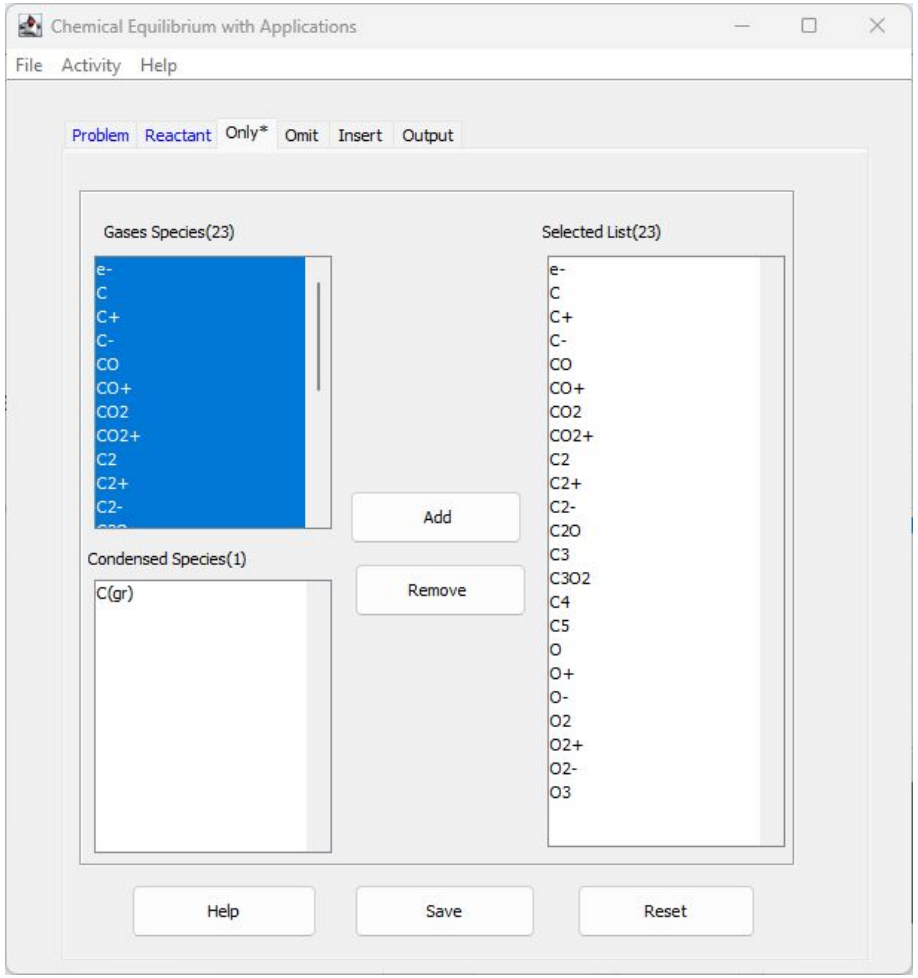




1 Data generation

Only tag / Omit / Insert

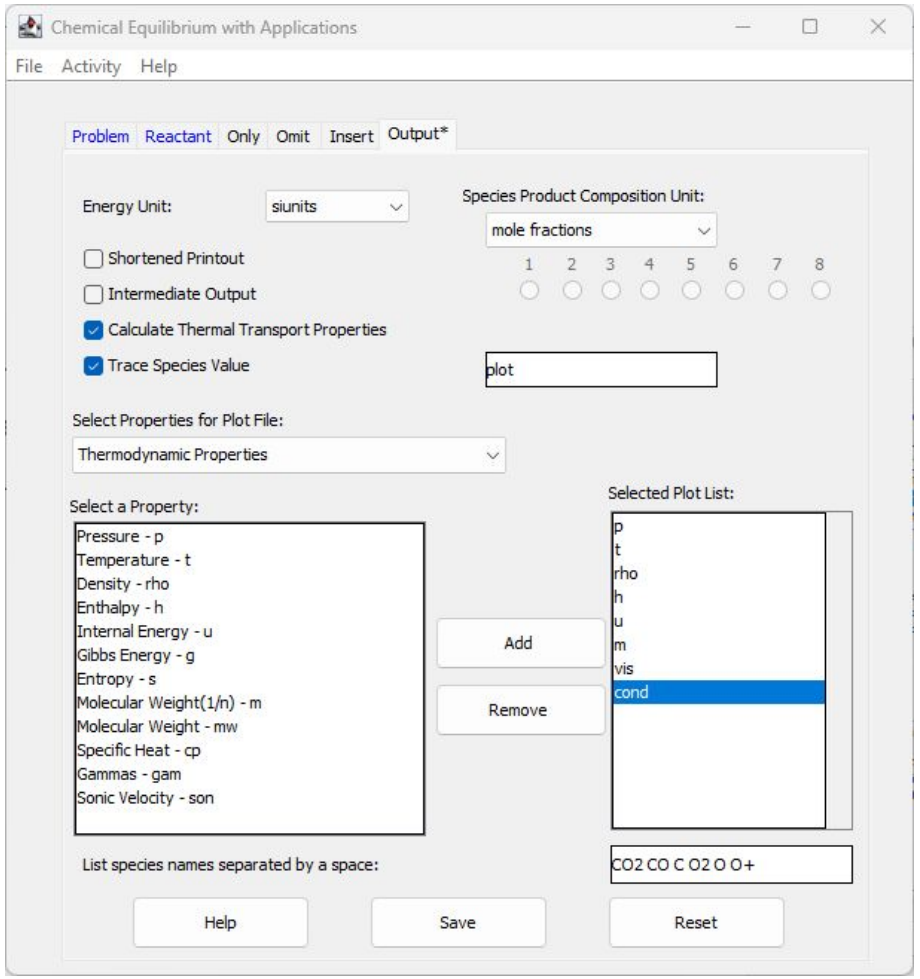
- If only a part of gases/ions are important for calculations, they can be specified in the **Only** tag. by clicking on them and pushing the **Add** button.
- The opposite happens for the **Omit** tag If there are gases that need to be excluded from computations.
- If solid particles play a considerable role in the calculations, it is possible to add them in the **Insert** tag.



1 Data generation

Output

- Check **Calculate Thermal Transport Properties** option if transport properties are desired such as Thermal conductivity, Viscosity, ..
- Select the desired gas properties and push add button.
- List the species name in the box separated by space in the bottom right box.

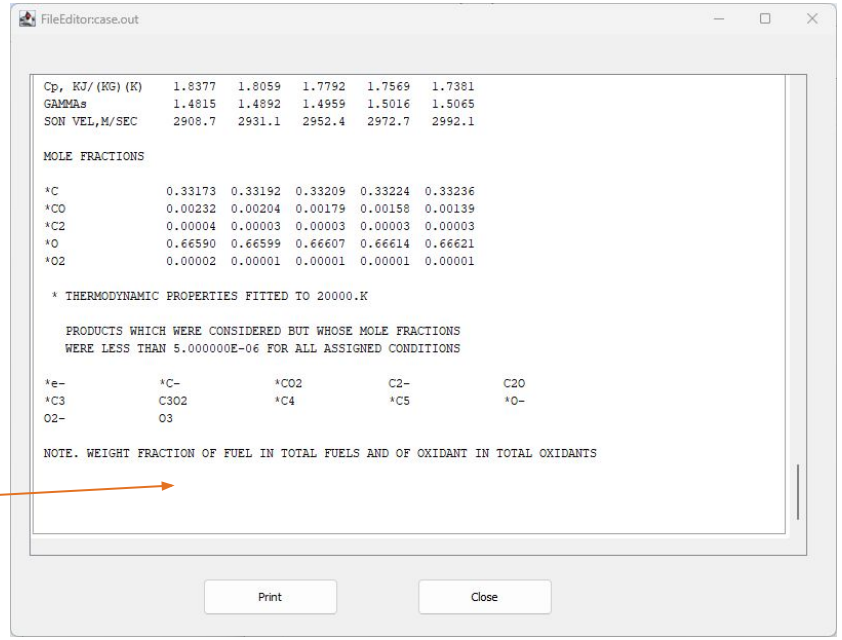


1 Data generation

Run program

- Run the program:
  - Select: `Activity/Execute CEA2`
  - Or push: `Ctrl+E`
- A successful execution does not show any error in the message box

No Error Here!

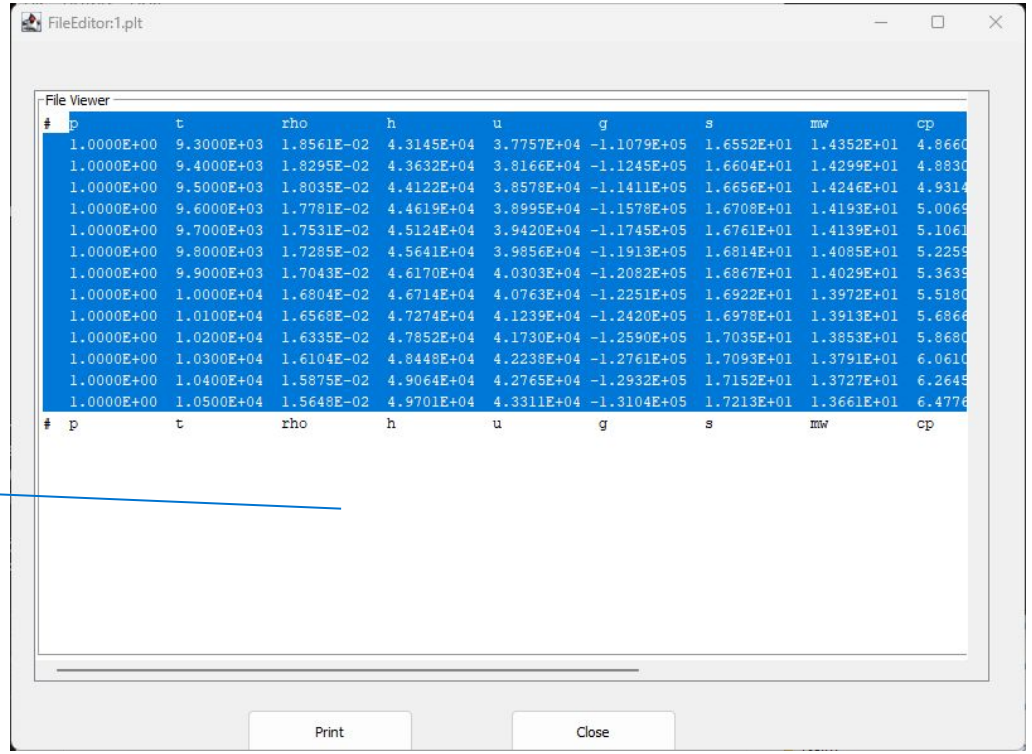


1 Data generation

Copy results

- See the results
  - Select: `Activity/View Plot file`
  - Or push: `Ctrl+D`
- copy the results and paste them into a excel or google sheet

A	B	C	D	E	F	G	H	I	J	K	L	M
p	t	rho	h	u	g	s	mw	cp	son	vis	cond	pran
2.00E-01	2.00E+02	5.29E-01	-9.02E+03	-9.06E+03	-9.99E+03	4.85E+00	4.40E+01	7.35E-01	2.26E+02	1.01E-01	9.55E-02	7.76E-01
2.00E-01	3.00E+02	3.53E-01	-8.94E+03	-9.00E+03	-1.05E+04	5.17E+00	4.40E+01	8.46E-01	2.70E+02	1.50E-01	1.66E-01	7.64E-01
2.00E-01	4.00E+02	2.65E-01	-8.85E+03	-8.93E+03	-1.10E+04	5.42E+00	4.40E+01	9.39E-01	3.08E+02	1.97E-01	2.46E-01	7.53E-01
2.00E-01	5.00E+02	2.12E-01	-8.75E+03	-8.85E+03	-1.16E+04	5.64E+00	4.40E+01	1.01E+00	3.41E+02	2.40E-01	3.26E-01	7.48E-01
2.00E-01	6.00E+02	1.76E-01	-8.65E+03	-8.76E+03	-1.21E+04	5.83E+00	4.40E+01	1.08E+00	3.71E+02	2.80E-01	4.03E-01	7.46E-01
2.00E-01	7.00E+02	1.51E-01	-8.54E+03	-8.67E+03	-1.27E+04	6.00E+00	4.40E+01	1.13E+00	3.99E+02	3.17E-01	4.78E-01	7.46E-01
2.00E-01	8.00E+02	1.32E-01	-8.42E+03	-8.57E+03	-1.33E+04	6.15E+00	4.40E+01	1.17E+00	4.25E+02	3.51E-01	5.48E-01	7.48E-01
2.00E-01	9.00E+02	1.18E-01	-8.30E+03	-8.47E+03	-1.40E+04	6.29E+00	4.40E+01	1.20E+00	4.49E+02	3.83E-01	6.16E-01	7.49E-01
2.00E-01	1.00E+03	1.06E-01	-8.18E+03	-8.37E+03	-1.46E+04	6.42E+00	4.40E+01	1.23E+00	4.72E+02	4.13E-01	6.80E-01	7.50E-01
2.00E-01	1.10E+03	9.62E-02	-8.06E+03	-8.27E+03	-1.53E+04	6.54E+00	4.40E+01	1.26E+00	4.94E+02	4.42E-01	7.41E-01	7.52E-01
2.00E-01	1.20E+03	8.82E-02	-7.93E+03	-8.16E+03	-1.59E+04	6.65E+00	4.40E+01	1.28E+00	5.16E+02	4.70E-01	8.01E-01	7.52E-01
2.00E-01	1.30E+03	8.14E-02	-7.80E+03	-8.05E+03	-1.66E+04	6.76E+00	4.40E+01	1.30E+00	5.36E+02	4.97E-01	8.61E-01	7.52E-01
2.00E-01	1.40E+03	7.56E-02	-7.67E+03	-7.93E+03	-1.73E+04	6.85E+00	4.40E+01	1.33E+00	5.55E+02	5.23E-01	9.26E-01	7.50E-01
2.00E-01	1.50E+03	7.06E-02	-7.54E+03	-7.82E+03	-1.80E+04	6.95E+00	4.40E+01	1.37E+00	5.74E+02	5.48E-01	1.00E+00	7.46E-01
2.00E-01	1.60E+03	6.61E-02	-7.40E+03	-7.70E+03	-1.87E+04	7.04E+00	4.40E+01	1.43E+00	5.91E+02	5.73E-01	1.11E+00	7.38E-01
2.00E-01	1.70E+03	6.22E-02	-7.25E+03	-7.57E+03	-1.94E+04	7.13E+00	4.39E+01	1.52E+00	6.08E+02	5.97E-01	1.26E+00	7.25E-01
2.00E-01	1.80E+03	5.86E-02	-7.09E+03	-7.43E+03	-2.01E+04	7.22E+00	4.38E+01	1.68E+00	6.24E+02	6.21E-01	1.48E+00	7.07E-01
2.00E-01	1.90E+03	5.53E-02	-6.91E+03	-7.27E+03	-2.08E+04	7.31E+00	4.37E+01	1.92E+00	6.39E+02	6.45E-01	1.81E+00	6.87E-01
2.00E-01	2.00E+03	5.23E-02	-6.70E+03	-7.08E+03	-2.15E+04	7.42E+00	4.34E+01	2.27E+00	6.54E+02	6.69E-01	2.28E+00	6.68E-01
2.00E-01	2.10E+03	4.93E-02	-6.45E+03	-6.86E+03	-2.23E+04	7.54E+00	4.31E+01	2.74E+00	6.71E+02	6.94E-01	2.92E+00	6.52E-01
2.00E-01	2.20E+03	4.65E-02	-6.15E+03	-6.58E+03	-2.31E+04	7.68E+00	4.25E+01	3.36E+00	6.89E+02	7.20E-01	3.77E+00	6.41E-01
2.00E-01	2.30E+03	4.37E-02	-5.77E+03	-6.23E+03	-2.38E+04	7.85E+00	4.18E+01	4.12E+00	7.09E+02	7.47E-01	4.84E+00	6.37E-01
2.00E-01	2.40E+03	4.09E-02	-5.32E+03	-5.81E+03	-2.46E+04	8.04E+00	4.08E+01	5.01E+00	7.32E+02	7.76E-01	6.09E+00	6.38E-01
2.00E-01	2.50E+03	3.81E-02	-4.77E+03	-5.30E+03	-2.54E+04	8.27E+00	3.96E+01	5.98E+00	7.58E+02	8.05E-01	7.48E+00	6.44E-01



# p	t	rho	h	u	g	s	mw	cp
1.0000E+00	9.3000E+03	1.8561E-02	4.3145E+04	3.7757E+04	-1.1079E+05	1.6552E+01	1.4352E+01	4.8660
1.0000E+00	9.4000E+03	1.8295E-02	4.3632E+04	3.8166E+04	-1.1245E+05	1.6604E+01	1.4299E+01	4.8830
1.0000E+00	9.5000E+03	1.8035E-02	4.4122E+04	3.8578E+04	-1.1411E+05	1.6656E+01	1.4246E+01	4.9314
1.0000E+00	9.6000E+03	1.7781E-02	4.4619E+04	3.8995E+04	-1.1578E+05	1.6708E+01	1.4193E+01	5.0069
1.0000E+00	9.7000E+03	1.7531E-02	4.5124E+04	3.9420E+04	-1.1745E+05	1.6761E+01	1.4139E+01	5.1061
1.0000E+00	9.8000E+03	1.7285E-02	4.5641E+04	3.9856E+04	-1.1913E+05	1.6814E+01	1.4085E+01	5.2259
1.0000E+00	9.9000E+03	1.7043E-02	4.6170E+04	4.0303E+04	-1.2082E+05	1.6867E+01	1.4029E+01	5.3639
1.0000E+00	1.0000E+04	1.6804E-02	4.6714E+04	4.0763E+04	-1.2251E+05	1.6922E+01	1.3972E+01	5.5180
1.0000E+00	1.0100E+04	1.6568E-02	4.7274E+04	4.1239E+04	-1.2420E+05	1.6978E+01	1.3913E+01	5.6866
1.0000E+00	1.0200E+04	1.6335E-02	4.7852E+04	4.1730E+04	-1.2590E+05	1.7035E+01	1.3853E+01	5.8680
1.0000E+00	1.0300E+04	1.6104E-02	4.8448E+04	4.2238E+04	-1.2761E+05	1.7093E+01	1.3791E+01	6.0610
1.0000E+00	1.0400E+04	1.5875E-02	4.9064E+04	4.2765E+04	-1.2932E+05	1.7152E+01	1.3727E+01	6.2645
1.0000E+00	1.0500E+04	1.5648E-02	4.9701E+04	4.3311E+04	-1.3104E+05	1.7213E+01	1.3661E+01	6.4776
# p	t	rho	h	u	g	s	mw	cp

1 Data generation

Manage data

- Continue this process for the desired set of pressure conditions
- Example:

If the sets of temperature and pressure conditions are:

Temperature = [200, 300, …, 9900, 10000] (K)

Pressure = [0.2, 0.3, …, 0.8, 0.9, 1.0] (bar)

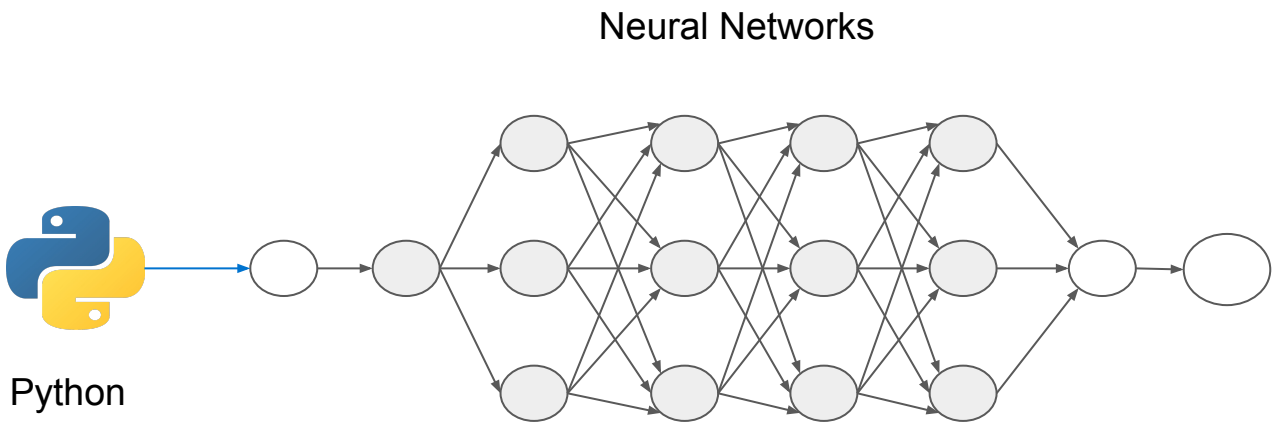
They need to be concatenated similar to the table.

- Save it as a *.csv file, and store in the python code directory.

  `/3d-data-fitting/data/dataset.csv`

Pressure (bar)	Temperature (K)	Density
0.2	200	
0.2	300	
0.2	…	
0.2	10000	
0.3	200	
0.3	300	
0.3	…	
0.3	10000	
…	…	
1.0	200	
1.0	300	
1.0	…	
1.0	10000	

2 Fit data with a model in python



2 Fit data with a model in python

Overall Process

- Install python and all necessary packages
- Read 3d Dataset
- Generate Neural network
- Train the model and extract the fitting parameters

2 Fit data with a model in python

Download & Install python and all necessary packages

Download **`3d-data-fitting`** python code from the github link:

- Link
- or use command: `git clone git@github.com:Erfan-Mashayekh/3d-data-fitting.git`

Make sure that the following packages and libraries are installed:

- **`Python 3`**
- **`Numpy`**: to work with arrays
- **`Matplotlib`**: to plot the results
- **`Tensorflow:`** to generate the neural network model
- **`Pandas`**: to read the csv file
- **`h5py`**: to save/load the trained model
- any **JSON** file reader library

2 Fit data with a model in python

Project Overview

Here is a short description of the files and directories in the repository: **`3d-data-fitting`**

- `main.py`: Coordinates and all functions.
- `manage_data.py`: Reads the settings and the dataset and manages it
- `model.py`: Defines the model and training strategy
- `plotter.py`: Plots the the dataset
- `utilities.py`: Contains necessary functions such dataset normalizers.
- `input.JSON`: This file controls the inputs and outputs and the methods
- `data`: The directory containing the dataset
- `output`: The directory that stores the model and the final parameters and figures

Modified by User

2 Fit data with a model in python

How to use `3d-data-fitting/data`

- copy the csv file in the `data` directory:
  - `/3d-data-fitting/data/dataset.csv`
- make sure in `manage.py` file, `read_properties()` function reads the correct file.
  - ```
    dataset = pd.read_csv('./data/dataset.csv')
    ```
    ```

2 Fit data with a model in python

How to use `3d-data-fitting/input.JSON`

- Open `input.json` file to control the settings:
- `train_mod:`
  - `“0”:` Parameter training is off. It is suitable for checking the data
  - `“1”:` Parameter training is on.
- `input_1 & input_2:` These are the two input values which are Temperature and pressure in this case.
- `output:` Type the name of the desired output property based on their given name in the dataset.
- `scale_input_1 & scale_input_2 & scale_output:` These are the scaling factors in case another unit is desired in fitting process. For instance, if the unit of specific heat capacity in the dataset is (kj/kg) but, for further steps, (j/kg) is desired, simply set the `scale_output` equal to 1000.

```
{
  "train_mod": "1",
  "input_1": "Temperature",
  "input_2": "Pressure",
  "plot_at_input_2": "0.2",
  "output": "Viscosity",
  "scale_input_1": "1.0",
  "scale_input_2": "1.0",
  "scale_output": "0.0001",
  "epochs": "5000",
  "loss": "mse",
  "optimizer": "adam",
  "metrics":"mse",
  "activation":"elu"
}
```

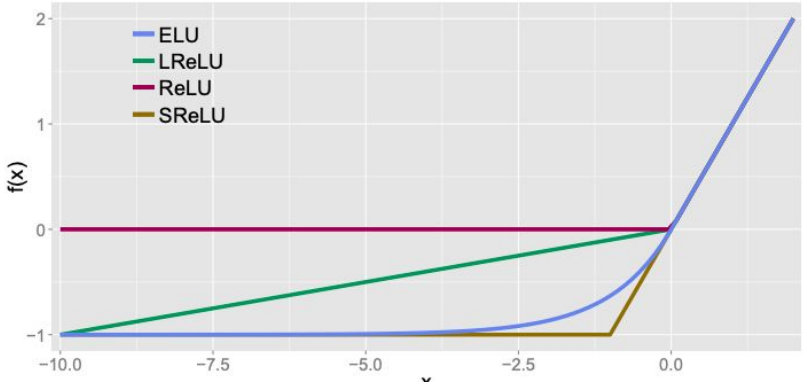
2 Fit data with a model in python

How to use `3d-data-fitting/input.JSON`

- `epochs:` Refers to the number of iterations for training.
- `loss:` Refers to the error or difference between the predicted output and the real data. Here, the mse method is selected as the loss function.
  - mse: Mean squared error (no need to change)

- `optimizer:` Refers to a module responsible for adjusting the model's parameters during training to minimize the loss function.
- `activation:` Activation function is a crucial component in a neural network, as it introduces non-linearity into the model, allowing it to learn and represent more complex relationships between input and output variables.
  - elu: For this model ELU function is selected.

```
{
  "train_mod": "1",
  "input_1": "Temperature",
  "input_2": "Pressure",
  "plot_at_input_2": "0.2",
  "output": "Viscosity",
  "scale_input_1": "1.0",
  "scale_input_2": "1.0",
  "scale_output": "0.0001",
  "epochs": "5000",
  "loss": "mse",
  "optimizer": "adam",
  "metrics":"mse",
  "activation":"elu"
}
```



2 Fit data with a model in python

How to run `3d-data-fitting`

- The package can be run by typing the following command in its repository:
  - `python3 ./main.py`
  - or open it in a python IDE and run `main.py` file

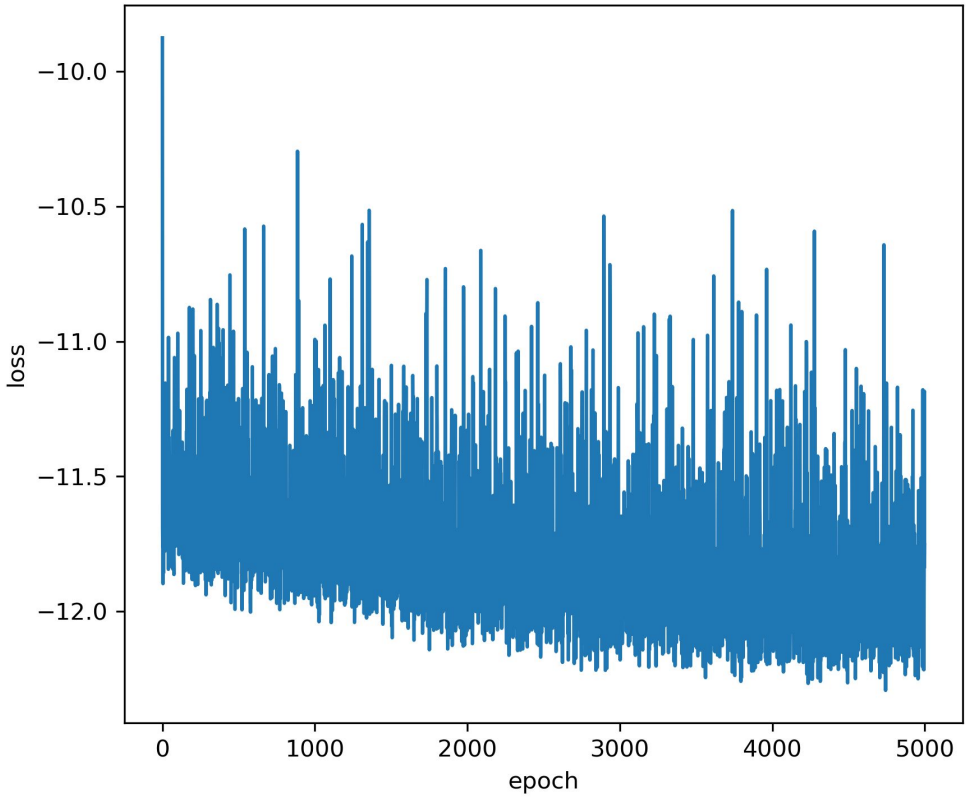
- After running the code the following files are stored in the `output` directory.
  - `loss.png:` Shows the training loss after iterations
  - `relative-error.png:` Shows the relative error between the fitted values and the real data
  - `solution-check.png:` Shows the difference between the fitted values and the real data graphs
  - `model.json:` Stores the structure of the model if further iteration is needed
  - `model.h5:` Stores the fitted parameters if further iteration is needed
  - `parameters.dat:` Stores the fitted parameters for copy and paste in the UDF (if needed)

2 Fit data with a model in python

Process the results

- `loss.png:` Shows the training loss after iterations

Based on experience loss values less than -7 or -8 proved to be in a good alignment with the real data

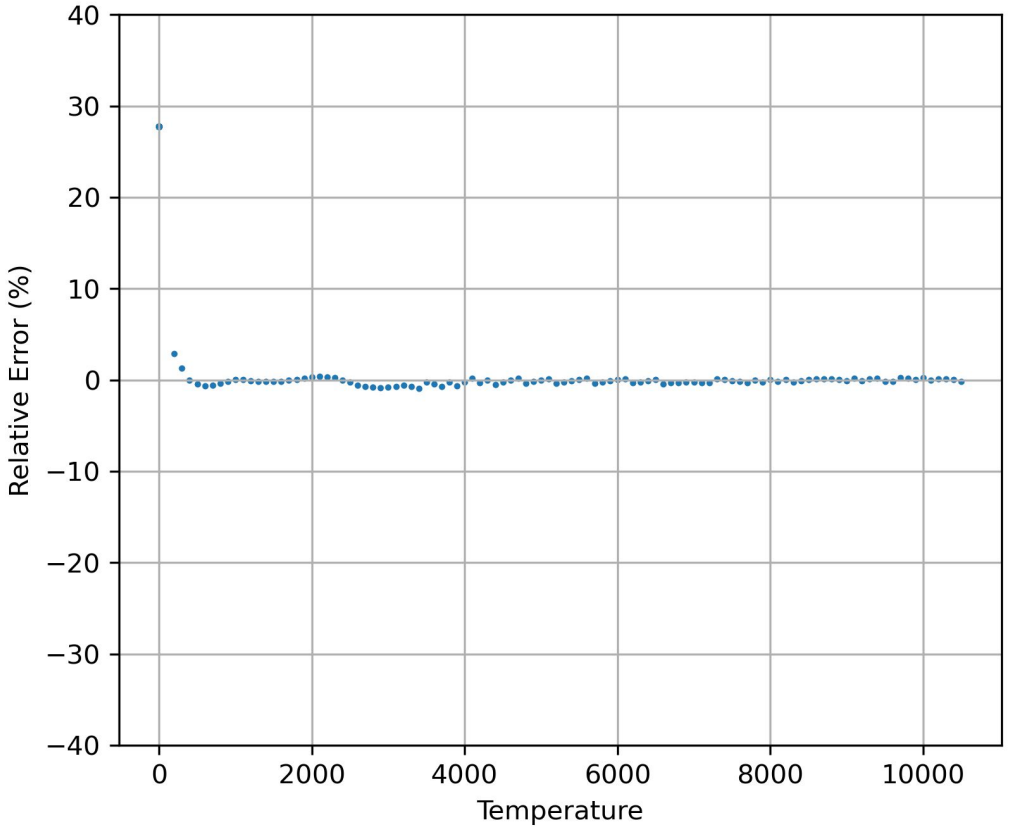


2 Fit data with a model in python

Process the results

- `relative-error.png:` Shows the relative error between the fitted values and the real data

Relative errors less than 10 percent showing a git fit.

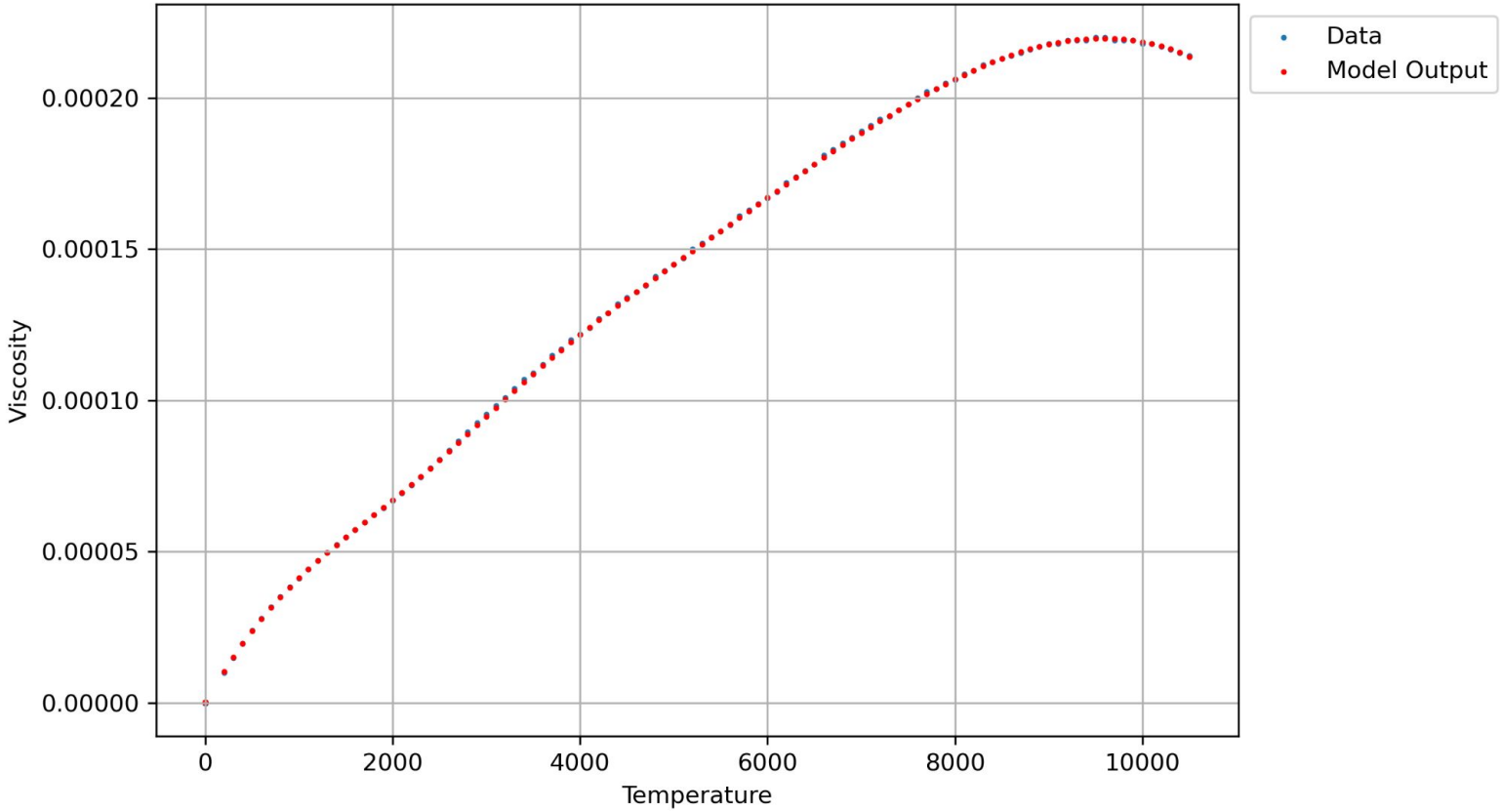


2 Fit data with a model in python

Process the results

- `solution-check.png:` Shows the difference between the fitted values and the real data graphs

If the model output does not follow the same curve as the real data, repeat the training process until it fits.



2 Fit data with a model in python

Process the results

- `parameters.dat:` Stores the fitted parameters for copy and paste in the UDF

This is the most important file. When training is finished, open this file and copy the necessary data in the UDF or C file for further processes. The details are explained in the next chapters.

```
Temperature: mean: 5350.0, std: 3003.687573206167
Pressure: mean: 0.599999999999997, std: 0.2583369271212281
Viscosity: mean: 0.00014446720085470072, std: 6.682687556213478e-05

 w1:

-0.07627664506435394
0.9941458106040955
-0.8704563975334167
-0.22618533670902252
-0.09883543848991394
0.16403548419475555

 b1:

0.41232216358184814
-0.8418604731559753
0.5989241600036621

 w2:

0.6636766195297241
-0.14462170004844666
0.16192249953746796
0.3613113760948181
-0.4963308274745941
1.1713380813598633
0.7856444120407104
-0.46768686175346375
0.4976082444190979

 b2:

0.037876639515161514
0.20873013138771057
-0.27797114849090576

 w3:

-1.1053541898727417
-0.22865509986877441
0.15506090223789215
0.6224076747894287
0.6232283115386963
0.5782488584518433
-0.2416839748620987
0.3030741214752197
0.680594801902771
0.05129415541887283
0.8046010136604309
0.8724228143692017

 b3:

0.056650690734386444
0.5072046518325806
```

3 Convert Python model to C (Optional)

3 Convert Python model to C

Overall Process

This chapter is optional, and is explained for who intend to check whether the model is converted correctly to C code. In order to find it, the following steps need to be done:

- Provide a C-based platform to execute the code.
- Copy the parameters from the parameter.dat file and paste it into the C-based model
- Run the simulations
- compare the results with the real data.

3 Convert Python model to C

Overall Process

The C-based code mimics the feed-forward process of the trained neural network. In order to use it for each property, we need to change its default parameters with the new trained parameters of the python step. In every C code, two parts need to be modified:

- Normalization step
- Weights and Biases

This information is available in the parameter.dat. To modify the C code, we need to update these lines with the information of the new trained parameters located in parameter.dat.

3 Convert Python model to C

Copy the parameters

- Normalization step

For each property, there is a line that normalize the Temperature, pressure and the desired property (for example, thermal conductivity). Simply copy the **mean** and standard deviation (**std**) for each property from the new obtained data located at the parameters.dat

C++ code for thermal conductivity

ThermalConductivity.cpp

```
double temperature = 0.0;
double temp_mean = 5350.0;
double temp_std = 3003.6875;

double pressure = 0.2;
double pressure_mean = 0.6;
double pressure_max = 1.0;
double pressure_std = 0.2583369;

double cond_mean = 1.3577453525641;
double cond_std = 1.1512988021;
```

Parameters.dat

```
Temperature: mean: 5350.0, std: 3003.687573206167
Pressure: mean: 0.6, std: 0.2583369271212277
Thermal Conductivity: mean: 1.3577453525641028,
std: 1.151298802100923

 w1:

-1.509173035621643
0.5237865447998047
2.913015365600586
```

3 Convert Python model to C

Copy the parameters

- Weights and Biases

Do the same process (copy & pasting) for the weights and biases. Repeat this process for all layers.

Each layer has a number that can be tracked.

C++ code for thermal conductivity

ThermalConductivity.cpp

```
if(layer == 1)
{
w[layer][0][0] = -1.509173035621643;
w[layer][1][0] = 0.5237865447998047;
w[layer][2][0] = 2.913015365600586;
w[layer][0][1] = 0.011024920269846916;
w[layer][1][1] = -0.7952345013618469;
w[layer][2][1] = -0.054956842213869095;

b[layer][0] = 0.07947462052106857;
b[layer][1] = -1.886416792869567;
b[layer][2] = -0.464243173599243;
...
}
```

Parameters.dat

```
Temperature: mean: 5350.0, std: 3003.687573206167
Pressure: mean: 0.6, std: 0.2583369271212277
Thermal Conductivity: mean: 1.3577453525641028,
std: 1.151298802100923

 w1:

-1.509173035621643
0.5237865447998047
2.913015365600586
0.011024920269846916
-0.7952345013618469
-0.054956842213869095

 b1:

0.07947462052106857
-1.8864167928695679
-0.46424317359924316

 w2:

0.9590182900428772
-0.5533522963523865
0.6885833740234375
1.214911937713623
0.39636021852493286
-0.9284956455230713
1.5494611263275146
```

Lehrstuhl für Energiesysteme
TUM School of Engineering and Design
Technische Universität München

3 Convert Python model to C

Run the C code

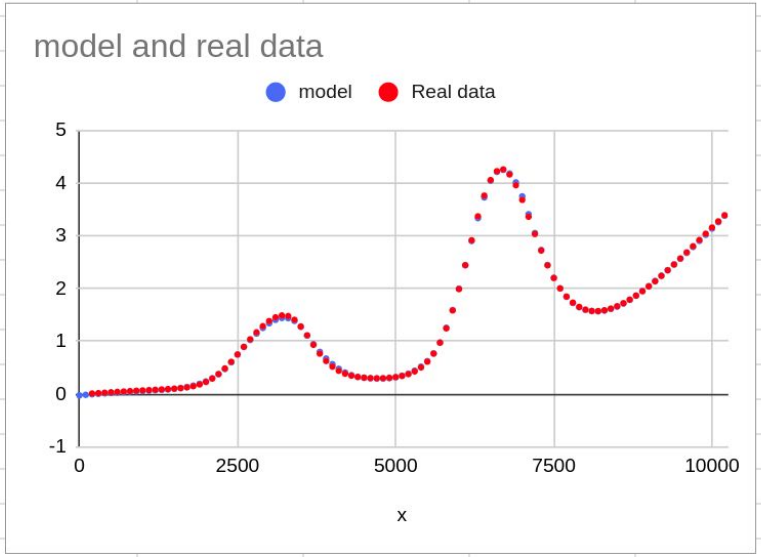
Now, it is time to run the C code and compare the results with the real data. Imagine that the simulation is done for thermal conductivity.

First, we need to set the pressure conditions at which we need to check the results. This can be done by setting it in the C code where the pressure is defined.

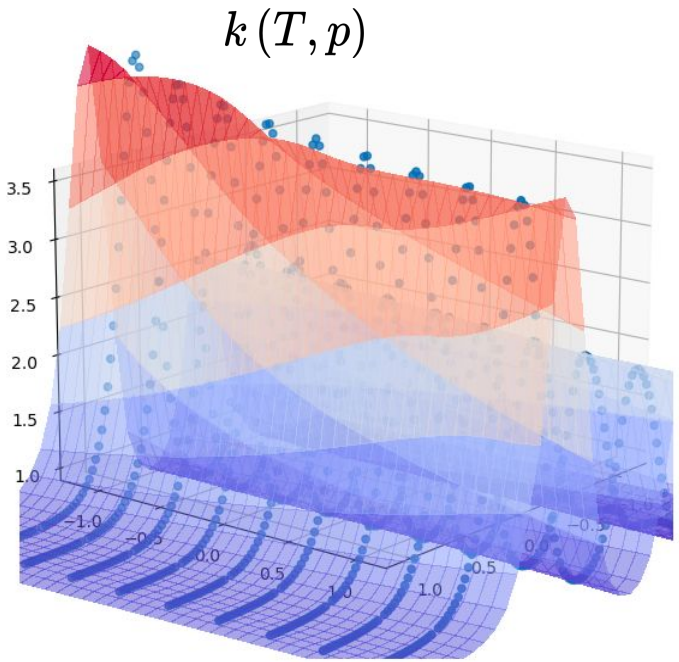
```
double pressure = 0.2;  //for 0.2 bar operating condition
```

Then compile and run the code. open the output directory In the C repository. Copy the data of `output-k.txt` (`k:` means thermal conductivity) and paste it in google sheet or excel to compare with real data.

temperature	model	Real data
0	-0.0276735	
100	-0.0157866	
200	-0.00560006	9.55E-03
300	0.00315323	1.66E-02
400	0.0114903	2.46E-02
500	0.0194739	3.26E-02
600	0.0271172	4.03E-02
700	0.0344483	4.78E-02
800	0.0415177	5.48E-02
900	0.0484084	6.16E-02
1000	0.0552522	6.80E-02
1100	0.0622523	7.41E-02
1200	0.0696326	8.01E-02
1300	0.0774052	8.61E-02
1400	0.0862075	9.26E-02
1500	0.0974644	1.00E-01
1600	0.11231	1.11E-01
1700	0.131978	1.26E-01
1800	0.158042	1.48E-01



4 Implement the UDF



FLUENT

4 Implement the UDF

Overall Process

This chapter elaborates on the steps for modifying the UDF file for the new parameters

Steps:

- Open the file `variablesproperty.c` located in the directory of FLUENT case&data files.
- Set the desired operating conditions at the top of the file:
  - #define Power 1500.0 // (optional) The overall power input in case of using microwave heat source (W).
  - #define Max_radius 0.0128 // (optional) this is to set the limit for the radius of the heat source in the reactor (m).
  - #define Axial_offset 0.031 // (optional) this to specify the starting axial position of the plasma (m).
  - #define Pressure 0.9 // Desired operating pressure condition (bar)

- Copy the parameters from the parameter.dat file and paste it into the C-based UDF. (explained in next slides)
- Save the file and compile it in FLUENT.

4 Implement the UDF

Copy the parameters

The UDF code contains the feed-forward process of the trained neural network. In order to use it for each property, we need to change its default parameters with the new trained parameters of the python step. Each property is defined using `DEFINE_PROPERTY(desired_property, c, t)` function. we need to update to lines at each of these functions:

- Normalization step
- Weights and Biases

This information is available in the parameter.dat. To modify the UDF, we need to update these lines with the information of the new trained parameters located in parameter.dat.

4 Implement the UDF

Copy the parameters

- Normalization step

For each property, there is a line that normalize the Temperature, pressure and the desired property (for example, thermal conductivity). Simply copy the **mean** and standard deviation (**std**) for each property with the new trained data located at the first lines of the parameters.dat

UDF file

DEFINE_PROPERTY(ThermalConductivity, c, t)

```
DEFINE_PROPERTY(thermalconductivity, c, t)
{
        const int numLayers = 7;
        const int maxNumNeurons = 4;;

        int row_w[numLayers], column_w[numLayers], row_l[numLaye
        double w[numLayers][maxNumNeurons][maxNumNeurons];
        double b[numLayers][maxNumNeurons];
        double output[numLayers][maxNumNeurons][maxNumNeurons];

        int layer, i, j, k;

        double temperature = C_T(c, t);
        double temp_mean = 5350.0;
        double temp_std = 3003.6875;

        double pressure_mean = 0.59999;
        double pressure_max = 1.0;
        double pressure_std = 0.2583369;

        double turb_thermal_conductivity = C_K_T(c, t, 0.85);
        double cond_mean = 1.3577453525641;

        double x, y, z;
```

Parameters.dat

```
Temperature: mean: 5350.0, std: 3003.687573206167
Pressure: mean: 0.6, std: 0.2583369271212277
Thermal Conductivity: mean: 1.3577453525641028,
std: 1.151298802100923

 w1:

-1.509173035621643
0.5237865447998047
2.913015365600586
```

4 Implement the UDF

Copy the parameters

- Weights and Biases

Do the same process of copy & pasting for the weights and biases. Repeat this process for all layers.

Each layer has a number that can be tracked.

UDF file

DEFINE_PROPERTY(ThermalConductivity, c, t)

```
// Note that weights  and biases are implemented in "
for (int layer = 1; layer < numLayers; layer++)
{
        if (layer == 1)
        {
                w[layer][0][0] = -1.50917303562164;
                w[layer][1][0] = 0.523786544799804;
                w[layer][2][0] = 2.91301536560058;
                w[layer][0][1] = 0.011024920269846916;
                w[layer][1][1] = -0.795234501361846;
                w[layer][2][1] = -0.0549568422138690;

                b[layer][0] = 0.0794746205210685;
                b[layer][1] = -1.886416792869567;
                b[layer][2] = -0.464243173599243;

                row_w[layer] = 3;
                column_w[layer] = 2;
                row_l[layer] = 3;
                column_l[layer] = 1;
        }
        else if (layer == 2)
        {
```

Parameters.dat

```
Temperature: mean: 5350.0, std: 3003.687573206167
Pressure: mean: 0.6, std: 0.2583369271212277
Thermal Conductivity: mean: 1.3577453525641028,
std: 1.151298802100923

 w1:

-1.509173035621643
0.5237865447998047
2.913015365600586
0.011024920269846916
-0.7952345013618469
-0.054956842213869095

 b1:

0.07947462052106857
-1.8864167928695679
-0.46424317359924316

 w2:

0.9590182900428772
-0.5533522963523865
0.6885833740234375
1.214911937713623
0.39636021852493286
-0.9284956455230713
1.5494611263275146
```

CEA model data Training

Erfan Mashayekh

Technische Universität München

Fakultät für Maschinenwesen

Lehrstuhl für Energiesysteme

Contact:

Email: mashayekh.e@gmail.com