M13-L2 Problem 1

Once more, we will study the stress prediction problem, this time using XGBoost, a very powerful boosting method.

```
import numpy as np
import matplotlib.pyplot as plt

import xgboost as xgb
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error


def plot_shape(dataset, index, model=None, lims=None):
    x = dataset["coordinates"][index][:,0]
    y = dataset["coordinates"][index][:,1]

    if model is None:
        c = dataset["stress"][index]
    else:
        c = model.predict(dataset["features"][index])

    if lims is None:
        lims = [min(c),max(c)]

    plt.scatter(x,y,s=5,c=c,cmap="jet",vmin=lims[0],vmax=lims[1])
    plt.colorbar(orientation="horizontal", shrink=.75,
pad=0,ticks=lims)
    plt.axis("off")
    plt.axis("equal")

def plot_shape_comparison(dataset, index, model, title=""):
    plt.figure(figsize=[6,3.2], dpi=120)
    plt.subplot(1,2,1)
    plot_shape(dataset,index)
    plt.title("Ground Truth",fontsize=9,y=.96)
    plt.subplot(1,2,2)
    c = dataset["stress"][index]
    plot_shape(dataset, index, model, lims = [min(c), max(c)])
    plt.title("Prediction",fontsize=9,y=.96)
    plt.suptitle(title)
    plt.show()

def load_dataset(path):
    dataset = np.load(path)
    coordinates = []
    features = []
    stress = []
```

```
    N = np.max(dataset[:,0].astype(int)) + 1
    split = int(N*.8)
    for i in range(N):
        idx = dataset[:,0].astype(int) == i
        data = dataset[idx,:]
        coordinates.append(data[:,1:3])
        features.append(data[:,3:-1])
        stress.append(data[:,-1])
    dataset_train = dict(coordinates=coordinates[:split],
features=features[:split], stress=stress[:split])
    dataset_test = dict(coordinates=coordinates[split:],
features=features[split:], stress=stress[split:])
    X_train, X_test = np.concatenate(features[:split], axis=0),
np.concatenate(features[split:], axis=0)
    y_train, y_test = np.concatenate(stress[:split], axis=0),
np.concatenate(stress[split:], axis=0)
    return dataset_train, dataset_test, X_train, X_test, y_train,
y_test

def get_shape(dataset,index):
    X = dataset["features"][index]
    y = dataset["stress"][index]
    return X, y

def eval_model(model, verbose=False):
    pred_train = model.predict(X_train)
    pred_test = model.predict(X_test)
    mse_train = mean_squared_error(y_train, pred_train)
    mse_test = mean_squared_error(y_test, pred_test)
    if verbose:
        print(f"Train MSE = {mse_train:.2e}")
        print(f"Test  MSE = {mse_test:.2e}")
    return mse_train, mse_test
```

Loading the data

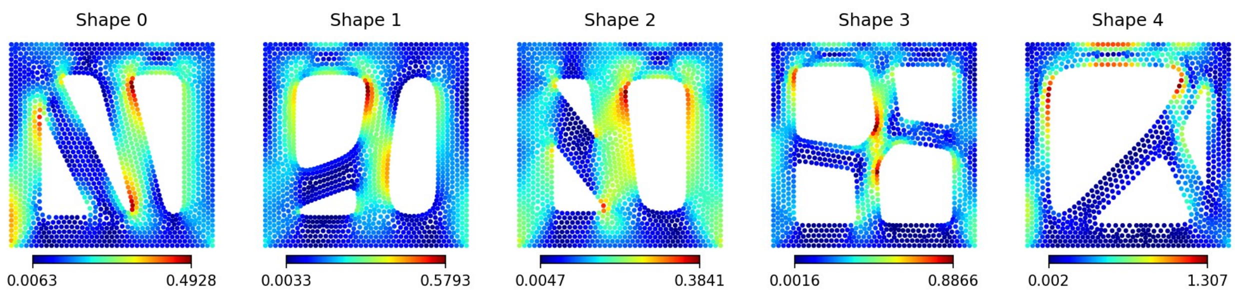
First, complete the code below to load the data and plot the von Mises stress fields for a few shapes.
You'll need to input the path of the data file, the rest is done for you.

All training node features and outputs are in `X_train` and `y_train`, respectively. Testing nodes are in `X_test`, `y_test`.

`dataset_train` and `dataset_test` contain more detailed information such as node coordinates, and they are separated by shape.
Get features and outputs for a shape by calling `get_shape(dataset,index)`. `N_train` and `N_test` are the number of training and testing shapes in each of these datasets.

```
# YOU MAY NEED TO EDIT data_path
data_path = "stress_nodal_features.npy"
dataset_train, dataset_test, X_train, X_test, y_train, y_test =
load_dataset(data_path)
N_train = len(dataset_train["stress"])
N_test = len(dataset_test["stress"])

plt.figure(figsize=[15,3.2], dpi=150)
for i in range(5):
    plt.subplot(1,5,i+1)
    plot_shape(dataset_train,i)
    plt.title(f"Shape {i}")
plt.show()
```



XGBoost Regressor

XGBoost models, like `XGBRegressor` here, can be used much like sklearn models.

First, define an instance of `XGBRegressor` with the desired parameters; then, fit the model with `model.fit`. You can evaluate a fitted model with `model.predict`.

The provided function `mse_train, mse_test = eval_model(model)` to get MSE values on the train and test datasets.

```
eta = 0.8
depth = 9

params = dict(
    eta = eta,
    max_depth = depth,
)

model = XGBRegressor(objective ='reg:squarederror', seed = 123,
n_estimators = 10, **params)
model.fit(X_train, y_train)

mse_train, mse_test = eval_model(model)
print("  eta   depth    |   Train MSE    Test MSE")
print("----------------|-------------------------")
```

```
print(f"  {eta:.1f}    {depth:>2d}     |    {mse_train:.2e}
{mse_test:.2e}")
```

```
  eta   depth    |   Train MSE    Test MSE
----------------|-------------------------
  0.8     9      |    2.19e-03    6.10e-03
```

Parametric study

Now let's examine the effects of varying the parameters `eta` and `max_depth`, keeping `n_estimators` as 10. For every combination of eta in [0.1, 0.3, 0.5, 0.7] and `max_depth` in [5, 10, 15, 20], train an XGB regressor and report the train and test MSE values.

Which combination has the best performance on testing data?

```
# YOUR CODE GOES HERE
eta = [0.1, 0.3, 0.5, 0.7]
depth = [5, 10, 15, 20]
print("  eta   depth    |   Train MSE    Test MSE")
print("----------------|-------------------------")

for e in eta:
    for d in depth:
        params = dict(
            eta = e,
            max_depth = d,
        )

        model = XGBRegressor(objective ='reg:squarederror', seed =
123, n_estimators = 10, **params)
        model.fit(X_train, y_train)

        mse_train, mse_test = eval_model(model)
        # print("  eta   depth    |   Train MSE    Test MSE")
        # print("----------------|-------------------------")
        print(f"  {e:.1f}    {d:>2d}     |    {mse_train:.2e}
{mse_test:.2e}")
        print("----------------|-------------------------")
```

```
  eta   depth    |   Train MSE    Test MSE
----------------|-------------------------
  0.1     5      |    1.05e-02    1.28e-02
----------------|-------------------------
  0.1    10      |    5.93e-03    8.93e-03
----------------|-------------------------
  0.1    15      |    3.86e-03    7.93e-03
----------------|-------------------------
  0.1    20      |    3.35e-03    7.98e-03
----------------|-------------------------
```

```
  0.3     5       |     5.43e-03    7.05e-03
----------------|--------------------------
  0.3    10       |     1.57e-03    4.58e-03
----------------|--------------------------
  0.3    15       |     2.79e-04    4.59e-03
----------------|--------------------------
  0.3    20       |     8.02e-05    4.73e-03
----------------|--------------------------
  0.5     5       |     4.60e-03    6.49e-03
----------------|--------------------------
  0.5    10       |     1.36e-03    4.80e-03
----------------|--------------------------
  0.5    15       |     1.47e-04    5.01e-03
----------------|--------------------------
  0.5    20       |     8.30e-06    5.20e-03
----------------|--------------------------
  0.7     5       |     4.83e-03    7.03e-03
----------------|--------------------------
  0.7    10       |     1.44e-03    5.54e-03
----------------|--------------------------
  0.7    15       |     1.60e-04    5.82e-03
----------------|--------------------------
  0.7    20       |     9.05e-06    6.02e-03
----------------|--------------------------
```

When eta = 0.3 and depth = 10, the model performed the best on the given testing data.