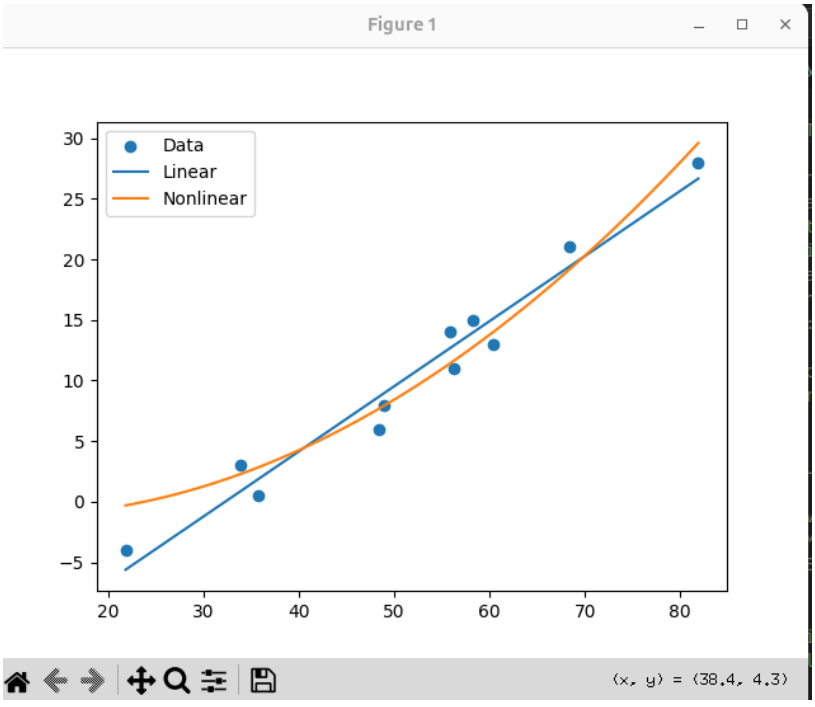
Problem 1: Nonlinear Temperature Forecasting

For the first problem, the original linear model was modified to accommodate a nonlinear system defined by the equation $t_p = w2 \times t_u^2 + w1 \times t_u + b$. The new parameter w2 was added, initialized to zero, and the training loop was updated to compute the corresponding gradient: $dw2 = 2 \times \text{mean}((t_p - t_c) \times t_u^2)$. To avoid numerical overflow, the input was normalized with $t_un = 0.1 \times t_u$, which greatly increased the stability during training. Training was performed on 5000 epochs with four different learning rates: 0.1, 0.01, 0.001, and 0.0001. Learning rates of 0.1 and 0.01 immediately diverged and returned NaN losses due to an overflow while computing the gradient. A learning rate of 0.001 had very large loss values in the beginning and then diverged to infinity at epoch 1500. Only a learning rate of 0.0001 converged, and its loss was steadily decreasing from 7.0097 at epoch 500 to 3.7775 at epoch 5000. This model was chosen as the best nonlinear variant.

The nonlinear model, therefore, compared poorly with the baseline linear model that was trained with a learning rate of 0.01, which had a final loss of 2.9276. Even though a quadratic term was added, a look into the temperature data shows that it is mainly linear; hence, adding a second-order term only added unnecessary complexity with little to no impact on the quality of fit. Such was further confirmed visually when both models were plotted over the input dataset: the predictions traced a line very close to the data points for the linear model and a slight parabolic curve for the nonlinear model, failing to capture the relationship from the true fit. In fact, the result was worse than the baseline linear model.



Problem 2: Linear Regression on Housing Dataset

In this project of housing price prediction, five input features were utilized: area, bedrooms, bathrooms, stories, and parking. The data was split into 80% training and 20% validation sets, and both the feature and target values were standardized using the mean and standard deviation from the training set for stable gradient descent. A linear model would be represented as $U = W5 \times X5 + W4 \times X4 + W3 \times X3 + W2 \times X2 + W1 \times X1 + B$ with six parameters, all initialized to zero. Training was conducted for 5000 epochs using the same four learning rates. It showed that learning rates 0.1 and 0.01 converged almost instantly to a training loss of 0.4378 and validation R^2 of 0.5464, which is indicative that big step sizes are enough to quickly reach the optimum. Learning rate 0.001 had more gradual convergence but finally came to the same final performance at epoch number 5000. In contrast, learning rate 0.0001 became too slow and finished with a training loss of 0.4621 and a validation R^2 of 0.4999. The best linear model could therefore be any with a learning rate of 0.001 or higher, which all achieved a validation R^2 of about 0.5464, meaning that about 54.6% of the variance in house prices is explained by the chosen features.

Problem 3:

Neural Networks on Housing Dataset First, a fully connected neural network was implemented with one hidden layer containing 8 nodes, all with ReLU activation. With the same 80/20 train-validation split and data normalization, this was trained for 200 epochs using SGD with a learning rate of 0.01. The training time was approximately 0.07 seconds, with a final training loss of 0.5685 and validation R^2 of 0.3876. This was significantly worse than the performance of the linear model, indicating that this model underfitted the data due to limited model capacity and/or not being trained for a sufficient amount of time. The network was then extended to three hidden layers of 8 nodes each with ReLU activation, adding depth. This model, under identical training conditions, finished in 0.05 seconds and yielded a training loss of 0.9986 and a validation R^2 of -0.0194. The negative R^2 indicates that the performance of the model is worse than a horizontal line (which would predict mean), which means it completely failed to learn any meaningful pattern. This could be due to vanishing gradients, bad weight initialization, or instability of full-batch SGD for a deeper network and relatively high learning rate. Relative to the linear baseline from Problem 2, both these neural networks performed worse, and despite having more parameters, the three-layer one severely overfitted or had training collapse. No improvement in accuracy was meaningful, and the added complexity was unstable without benefit. Future improvements could include using the Adam optimizer, mini-batch training,

weight initialization, or early stopping in order to better exploit nonlinear modeling capacity.