Report for Exercise 1

Daniel Zügner (6832915), Konstantin Kobs (6414943), Sazan Hoti (6832974)

The best fitting model to the generated data points using Stochastic Gradient Descent turned out to be:

$$f(x) = 0.22 + 5.05x - 9.50x^2 - 5.19x^3 + 1.97x^4 + 7.92x^5$$

We implemented the algorithm in MATLAB so that the degree of the polynomial can be set. The coefficients above were found for a degree of five. Note that the Θ values are rounded to the second decimal value. The data values obtained with this model are visualized in Figure 1.

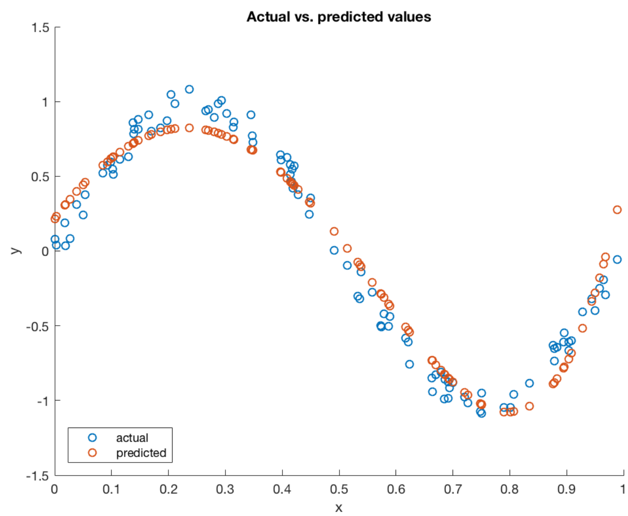

Figure 1: Shows the randomly generated data points (blue circles) in comparison to the predicted y values (red circles) for the same x values.

The above final model was obtained by setting the learning rate $\alpha = 0.001$. This led to 39426 iterations of the algorithm. Figure 2 shows that increasing α values lead to higher residual square error, however, increasing values of α also lead to faster convergence (see Figure 3).

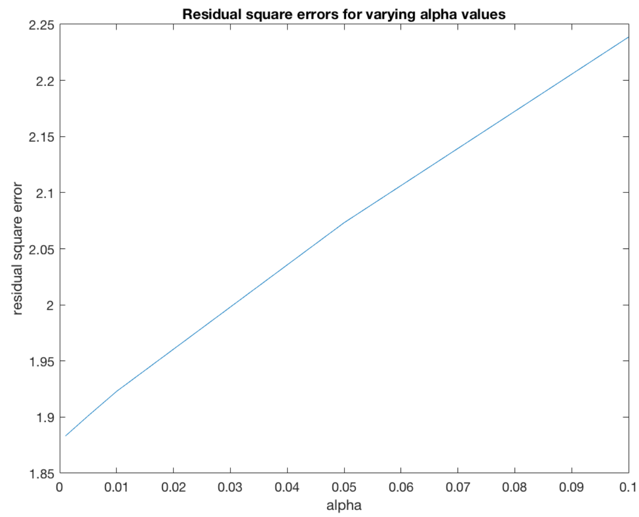

Figure 2: Shows the residual square error of the model with respect to increasing α values.

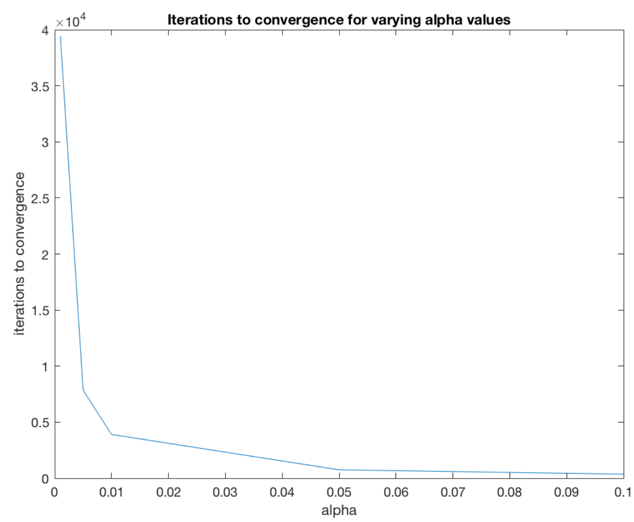

Figure 3: Shows the number of iterations the algorithm needed to converge depending on the α value.

Source code

In the following the main implementation of Stochastic Gradient Descent for this polynomial regression written in MATLAB.

```
function theta = stochasticGradientDescent (data, polynomial, alpha)
    display(['Polynomial: ' num2str(polynomial) ' Alpha: ' num2str(alpha)]);

    % initialize
    theta = rand(1,polynomial + 1);
    exponents = [0:polynomial];

    [rows, columns] = size(data);
    threshold = 0.01;
    update = ones(1,polynomial + 1);
    errors = [];

    iteration = 0;
    allUpdates = ones(1,polynomial + 1);
    errorDecrease = 1;
    while (errorDecrease / alpha > threshold)
        error = 0;
        allUpdates = zeros(1,polynomial + 1);
        for j = 1:rows
            element = data(j,:);
            x = element(1);
            y = element(2);
            xnew = x.^exponents;


            h = xnew * theta';
            error = error + (h - y).^2;

            update = alpha * (y - h) * xnew;
            allUpdates = allUpdates + update;
            theta = theta + update;

        end
        if iteration > 0
            errorDecrease = ((errors(end) - error)/errors(end));
        end
        iteration = iteration + 1;
        errors = [errors error];
    end
    figure
    plot([1: iteration], errors);
    title('Error values');
    xlabel('iteration');
    ylabel('square error');
    display(['Converged after ' num2str(iteration) ' iterations.']);
    display(['Resulting theta: ' mat2str(theta)]);
```

```
    display(['Summed residual square error: ' num2str(errors(end))]);
end
```