CS340 - Assignment 3

Tristan Rice - q7w9a - 25886145

1. Vectors, Matrices, and Quadratic Functions

1.1. Basic Operations

1.1.1.

$$x^T x = 2 * 2 + 3 * 3 = 13$$

1.1.2.

$$\|x\|^2 = x^T x = 13$$

1.1.3.

$$x^T(x + \alpha y) = \begin{bmatrix} 2 & 3 \end{bmatrix} \begin{bmatrix} 2 + 5 * 1 \\ 3 + 5 * 4 \end{bmatrix} = 2 * (2 + 5 * 1) + 3(3 + 5 * 4) = 83$$

1.1.4.

$$Ax = \begin{bmatrix} 2 * 1 + 3 * 2 \\ 2 * 2 + 3 * 3 \\ 3 * 3 + 3 * 2 \end{bmatrix} = \begin{bmatrix} 8 \\ 13 \\ 15 \end{bmatrix}$$

1.1.5.

$$z^T Ax = \begin{bmatrix} 2 & 0 & 1 \end{bmatrix} Ax = 2 * 8 + 0 * 13 + 1 * 15 = 31$$

1.1.6.

$$A^T A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 2 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 2 \end{bmatrix} = \begin{bmatrix} 1 * 1 + 2 * 2 + 3 * 3 & 1 * 2 + 2 * 3 + 3 * 2 \\ 1 * 2 + 2 * 3 + 3 * 2 & 2 * 2 + 3 * 3 + 2 * 2 \end{bmatrix} = \begin{bmatrix} 14 & 14 \\ 14 & 17 \end{bmatrix}$$

1.1.7. $yy^T y = \|y\|^2 y$

True. Matrix multiplication is associative. $y(y^T y) = (y^T y)y$. $y^T y$ produces a scalar, and scalar-matrix multiplication is commutative.

Alternate Way

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

$$\begin{bmatrix} y_1^2 & y_1 * y_2 & y_1 * y_3 \\ y_1 * y_2 & y_2^2 & y_2 * y_3 \\ y_1 * y_3 & y_2 * y_3 & y_3^2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = (y_1^2 + y_2^2 + y_3^2) \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

$$\begin{bmatrix} y_1^3 + y_1 * y_2^2 + y_1 * y_3^2) \\ y_1^2 * y_2 + y_2^3 + y_2 * y_3^2) \\ y_1^2 * y_3 + y_2^2 * y_3 + y_3^3) \end{bmatrix} = \begin{bmatrix} y_1^3 + y_1 * y_2^2 + y_1 * y_3^2) \\ y_1^2 * y_2 + y_2^3 + y_2 * y_3^2) \\ y_1^2 * y_3 + y_2^2 * y_3 + y_3^3) \end{bmatrix}$$

1.1.8. $x^T A^T (Ay + Az) = x^T A^T Ay + z^T A^T Ax$

True. Multiplication is distributive.

$$x^T A^T Ay + x^T A^T Az = x^T A^T Ay + z^T A^T Ax$$

$A^T A$ produces a scalar which is communative and $z^T x = x^T z$.

1.1.9. $A^T (B + C) = BA^T + CA^T$

False. Matrix multiplication is distributive, but not communative.

$$A^T B + A^T C \neq BA^T + CA^T$$

1.1.10. $x^T (B + C) = Bx + Cx$

False. Order typically matters when doing matrix multiplication.

$$x^T B + x^T C = Bx + Cx$$

1.1.11. $(A + BC)^T = A^T + C^T B^T$

True.

$$A^T + (BC)^T = A^T + C^T$$
$$A^T + C^T B^T = A^T + C^T B^T$$

1.1.12. $(x - y)^T (x - y) = \|x\|^2 - x^T y + \|y\|^2$

False.

$$(x^T - y^T)(x - y)$$
$$(x^T - y^T)x - (x^T - y^T)y)$$
$$x^T x - y^T x - x^T y + y^T y)$$
$$\|x\|^2 - 2x^T y + \|y\|^2$$

1.1.13. $(x - y)^T (x + y) = \|x\|^2 - \|y\|^2$

True.

$$(x^T - y^T)(x + y)$$
$$(x^T - y^T)x + (x^T - y^T)y)$$
$$x^T x - y^T x + x^T y - y^T y)$$
$$\|x\|^2 - \|y\|^2$$

1.2

1.2.1. $\sum_{i=1}^{n} |w^T x_i - y_i| = w^T X - y$

$$[1, 1, \ldots, 1](w^T X - y)$$

1.2.2. $\max_{i\in\{1,2,\ldots,n\}} |w^T x_i - y_i| + \frac{\lambda}{2}\sum_{j=1}^{n} w_j^2$

$$\log([1,1,\ldots,1]\exp(w^T X - y)) + \frac{\lambda}{2}\|w\|^2$$

exp is component wise exponentiation.

1.2.3. $\sum_{i=1}^{n} z_i(w^T x_i - y_i)^2 + \lambda\sum_{j=1}^{d}|w_j|$

$$(Xw-y)^T Z(Xw-y) + \lambda[1,1,\ldots,1]|w|$$

1.3

1.3.1. $f(w) = \frac{1}{2}\|w-v\|^2$

$$\frac{df(w)}{dw_1} = \frac{1}{2}\frac{d}{dw_1}((w_1-v_1)^2 + (w_2-v_2)^2 + \ldots)$$
$$\frac{df(w)}{dw_1} = \frac{1}{2}2(w_1-v_1)$$
$$\frac{df(w)}{dw_1} = w_1 - v_1$$
$$\nabla f(w) = [\frac{df(w)}{dw_1},\ldots,\frac{df(w)}{dw_n}]$$

$$\nabla f(w) = w - v = 0$$
$$w = v$$

1.3.2 $f(w) = \frac{1}{2}\|w\|^2 + w^T X^T y$

$$f(w) = \frac{1}{2}\|w\|^2 + y^T X^T w$$
$$\nabla f(w) = w + y^T X^T = 0$$
$$w = -y^T X^T$$

1.3.3. $f(w) = \frac{1}{2}\|Xw-y\|^2 + \frac{1}{2}w^T\Lambda w$

$$\nabla f(w) = X(Xw-y) + \Lambda w) = 0$$
$$XXw - Xy + \Lambda w) = 0$$
$$(XX+\Lambda)w = Xy$$
$$w = (XX+\Lambda)^{-1}Xy$$

1.3.4 $f(w) = \frac{1}{2}\sum_{i=1}^{n} z_i(w^T x_i - y_i)^2$

$$f(x) = \frac{1}{2}(Xw - y)^T Z(Xw - y)$$

$$f(x) = \frac{1}{2}(w^T X^T - y^T)Z(Xw - y)$$

$$f(x) = \frac{1}{2}(w^T X^T - y^T)(ZXw - Zy)$$

$$f(x) = \frac{1}{2}((w^T X^T - y^T)ZXw - (w^T X^T - y^T)Zy)$$

$$f(x) = \frac{1}{2}(w^T X^T ZXw - y^T ZXw - w^T X^T Zy + y^T Zy)$$

$$\nabla f(x) = X^T ZXw - \frac{1}{2}y^T ZX - \frac{1}{2}X^T Zy = 0$$

$$X^T ZXw = \frac{1}{2}y^T ZX + \frac{1}{2}X^T Zy$$

$$w = \frac{1}{2}((X^T ZX)^{-1}y^T ZX + (X)^{-1}y)$$

Solving for y (Z=1)

$$X^T Xw = \frac{1}{2}y^T X + \frac{1}{2}X^T y$$

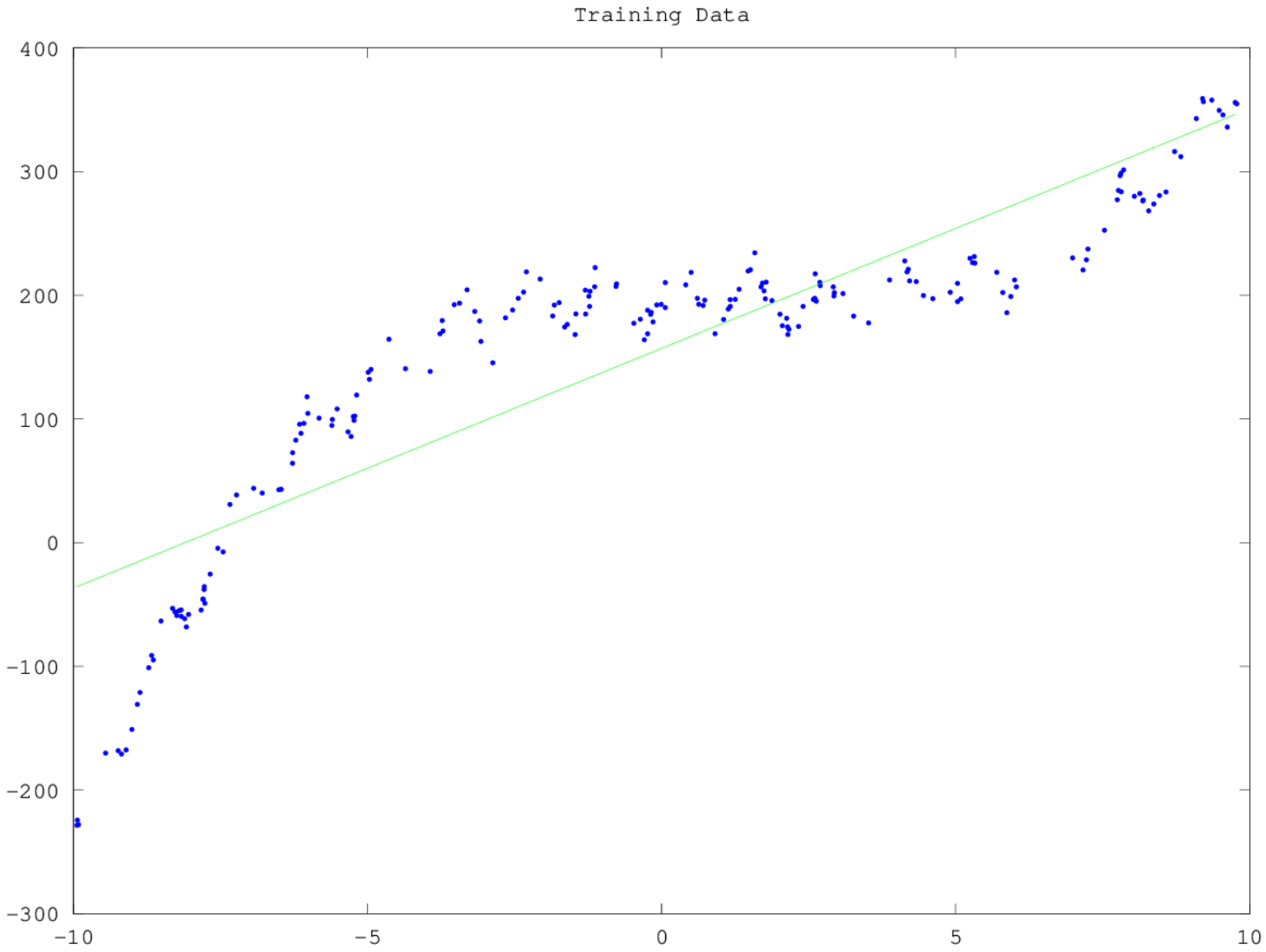
$$X^T Xw = \frac{1}{2}X^T y + \frac{1}{2}X^T y$$

$$Xw = y$$

2. Linear Regression and Nonlinear Bases

2.1. Adding a Bias Variable

```
function [model] = leastSquaresBias(X,y)

X = [ones(rows(X), 1) X];

% Solve least squares problem
w = (X'*X)\X'*y;

model.w = w;
model.predict = @predict;

end

function [yhat] = predict(model,Xhat)
Xhat = [ones(rows(Xhat), 1) Xhat];
w = model.w;
yhat = Xhat*w;
end
```




```
Training error = 3551.35
Test error = 3393.87
```

2.2. Polynomial Basis

```
function [model] = leastSquaresBasis(X,y,p)

X = polyBasis(X, p);

% Solve least squares problem
% We're using inv, since octave works better when dealing with large matrixes.
w = inv(X'*X)*X'*y;

model.w = w;
model.p = p;
model.predict = @predict;

end

function [yhat] = predict(model,Xhat)
Xhat = polyBasis(Xhat, model.p);
w = model.w;
yhat = Xhat*w;
```

```
end

function [Xpoly] = polyBasis(X, p)
Xpoly = [];
for i = 0:p
  Xpoly = [Xpoly X.^i];
end
end
```

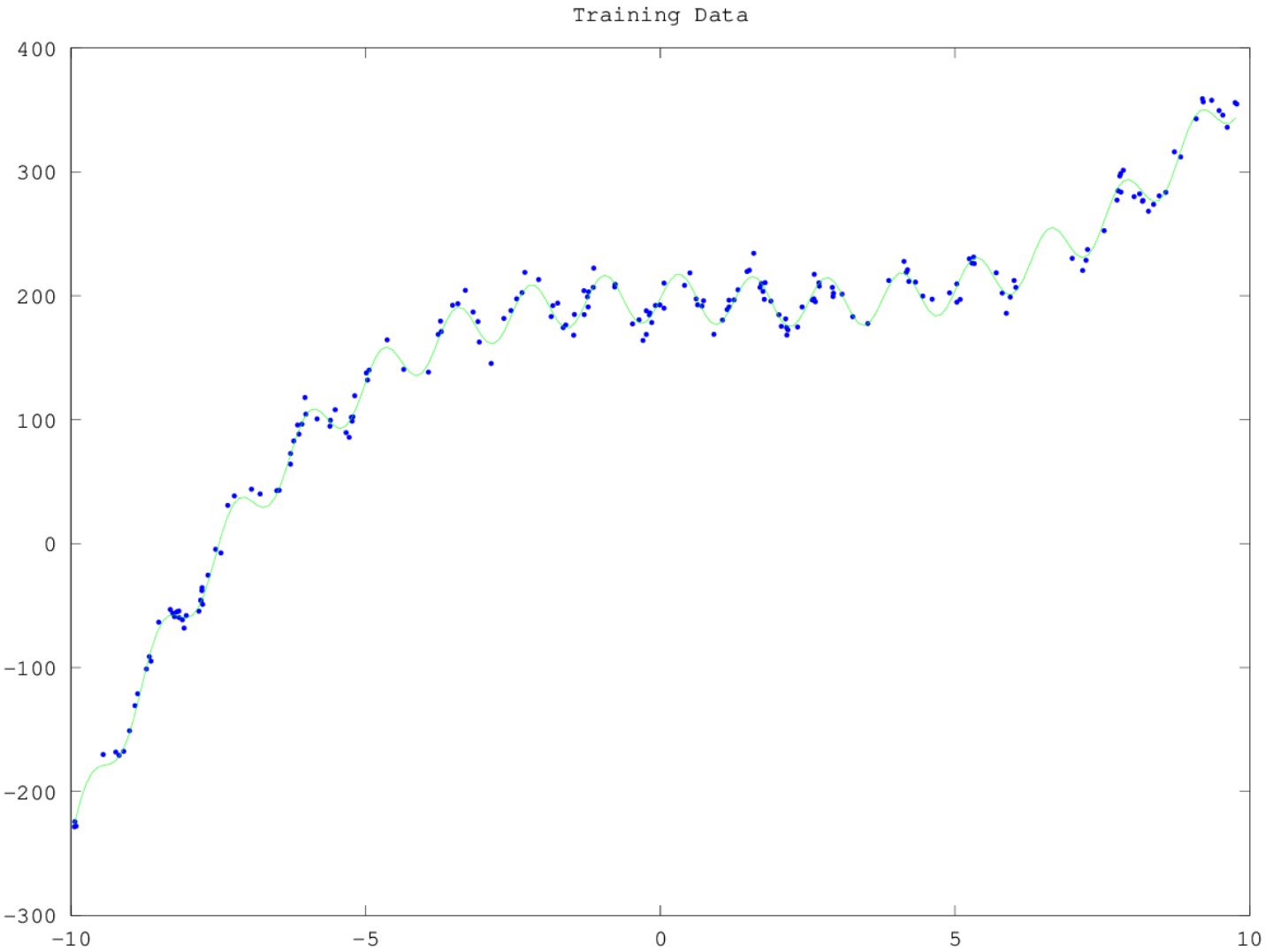
```
p = 0:
Training error = 15480.52
Test error = 14390.76
p = 1:
Training error = 3551.35
Test error = 3393.87
p = 2:
Training error = 2167.99
Test error = 2480.73
p = 3:
Training error = 252.05
Test error = 242.80
p = 4:
Training error = 251.46
Test error = 242.13
p = 5:
Training error = 251.14
Test error = 239.54
p = 6:
Training error = 248.58
Test error = 246.01
p = 7:
Training error = 247.01
Test error = 242.89
p = 8:
Training error = 241.31
Test error = 245.97
p = 9:
Training error = 235.76
Test error = 259.30
p = 10:
Training error = 235.07
Test error = 256.30
```

Increasing p lowers the training error, but doesn't necessarily lower the test error. With very high p values, the model becomes over fit and performs less well.

2.3. Manual Search for Optimal Basis

Fitting a $p = 5$ Least Squares Basis and then adding $20 * \sin(X * 5)$ gives us

```
Training error = 50.99
Test error = 53.71
```



This is a pretty good fit. However, from the naked eye, it looks like it could be a bit better.

I obtained this basis from using the best p value from the previous question, and then manually measuring the number of peaks. Manually counting the number of peaks and then estimating the height of them gives a pretty good estimate for the coefficients for the sin term.

3. Non-Parametric Bases and Cross-Validation

3.1. Proper Training and Validation Sets

The issue is that the training and validation sets aren't both representative of the whole dataset since they're respectively the first and second half of it. If you instead randomize the selection it works much better and is a better representation of the dataset.

```
ordering = randperm(n);
X = X(ordering,:);
y = y(ordering,:);
Xtrain = X(1:n/2,:);
ytrain = y(1:n/2,:);
Xvalid = X(n/2+1:end,:);
yvalid = y(n/2+1:end,:);
```

```
Training error = 39.49
Test error = 71.17
```

3.2. Cross-Validation

```
% Load data
load basisData.mat % Loads X and y
[n,d] = size(X);

% Split training data into a training and a validation set
ordering = randperm(n);
X = X(ordering,:);
y = y(ordering,:);

valsize = n/10;

bestSigmas = [];

for cv=1:10
    validation = ((cv-1)*valsize+1):((cv)*valsize);
    training = setdiff(1:n, validation);
    Xtrain = X(training,:);
    ytrain = y(training,:);
    Xvalid = X(validation,:);
    yvalid = y(validation,:);

    % Find best value of RBF kernel parameter,
    %   training on the train set and validating on the validation set
    minErr = inf;
    for sigma = 2.^[-15:15]

        % Train on the training set
        model = leastSquaresRBF(Xtrain,ytrain,sigma);

        % Compute the error on the validation set
        yhat = model.predict(model,Xvalid);
        validError = sum((yhat - yvalid).^2)/(n/2);
        fprintf('Error with sigma = %.3e = %.2f\n',sigma,validError);

        % Keep track of the lowest validation error
        if validError < minErr
            minErr = validError;
            bestSigma = sigma;
        end
    end
    fprintf('Value of sigma that achieved the lowest validation error was %.3e\n',bestSigma);
    bestSigmas = [bestSigmas bestSigma];
end

bestSigma = mean(bestSigmas)

% Now fit the model based on the full dataset.
fprintf('Refitting on full training set...\n');
model = leastSquaresRBF(X,y,bestSigma);

% Compute training error
yhat = model.predict(model,X);
trainError = sum((yhat - y).^2)/n;
fprintf('Training error = %.2f\n',trainError);

% Finally, report the error on the test set
t = size(Xtest,1);
yhat = model.predict(model,Xtest);
testError = sum((yhat - ytest).^2)/t;
```

```
fprintf('Test error = %.2f\n',testError);
```

Typical output

```
bestSigma =  0.95000
Refitting on full training set...
Training error = 39.16
Test error = 63.29
```

The best σ value is typically 0.95, but is sometimes 1 or 0.9.

3.3. Cost of Non-Parametric Bases

Training linear basis

Matlab code:

```
w = (X'*X)\X'*y;
```

`X'*X` is $O(d^2n)$.

`X'*y` is $O(dn)$.

Inversion of a $d \times d$ matrix is $O(d^3)$.

$d \times d$ matrix times a $d \times 1$ matrix is $O(d^2)$.

Thus, total complexity is $O(d^2n + dn + d^3 + d^2)$. Assuming $d < n$, we then get $O(d^2n)$.

Classification is just $\hat{X}w$ which is $O(t * d)$.

Training Gaussian RBF

From the `leastSquaresRBF.m` function.

Finding the rbf basis requires 3, $n_1 \times d * d \times n_2$ matrix multiplications, two $n \times d$ component wise squaring and some other $n_1 \times n_2$ component wise operations. Thus, finding the rbf basis is $O(n_1n_2d + n_1 * d + n_2 * d + n_1n_2) = O(n_1n_2d)$.

```
w = (Z'*Z + lambda*eye(n))\Z'*y;
```

When training $n_1 = n_2 = n$, thus $O(n^2d + n^3 + n^2) = n^3$. The most expensive step is inverting the $n \times n$ matrix which is $O(n^3)$.

```
yhat = Z*model.w;
```

For testing, creating the rbf basis is $O(ntd)$. w is a $n \times 1$ matrix, so Zw is $O(nt)$. Thus, the runtime of rbf for testing is $O(ntd)$.

Which is cheaper?

RBFs are cheaper to train when $d > n$ and are typically more expensive to test.

3.4. Non-Parametric Bases with Uneven Data

When the period of oscillations isn't constant, it's much harder to fit an RBFs to it since the optimal n will change. For high frequency sections, the RBF needs to be much closer together. When there are low frequency sections, the RBFs may be denser than required.

If we haven't evenly sampled the training data, there may be gaps or sections at either end of the domain that the RBFs don't have data for.

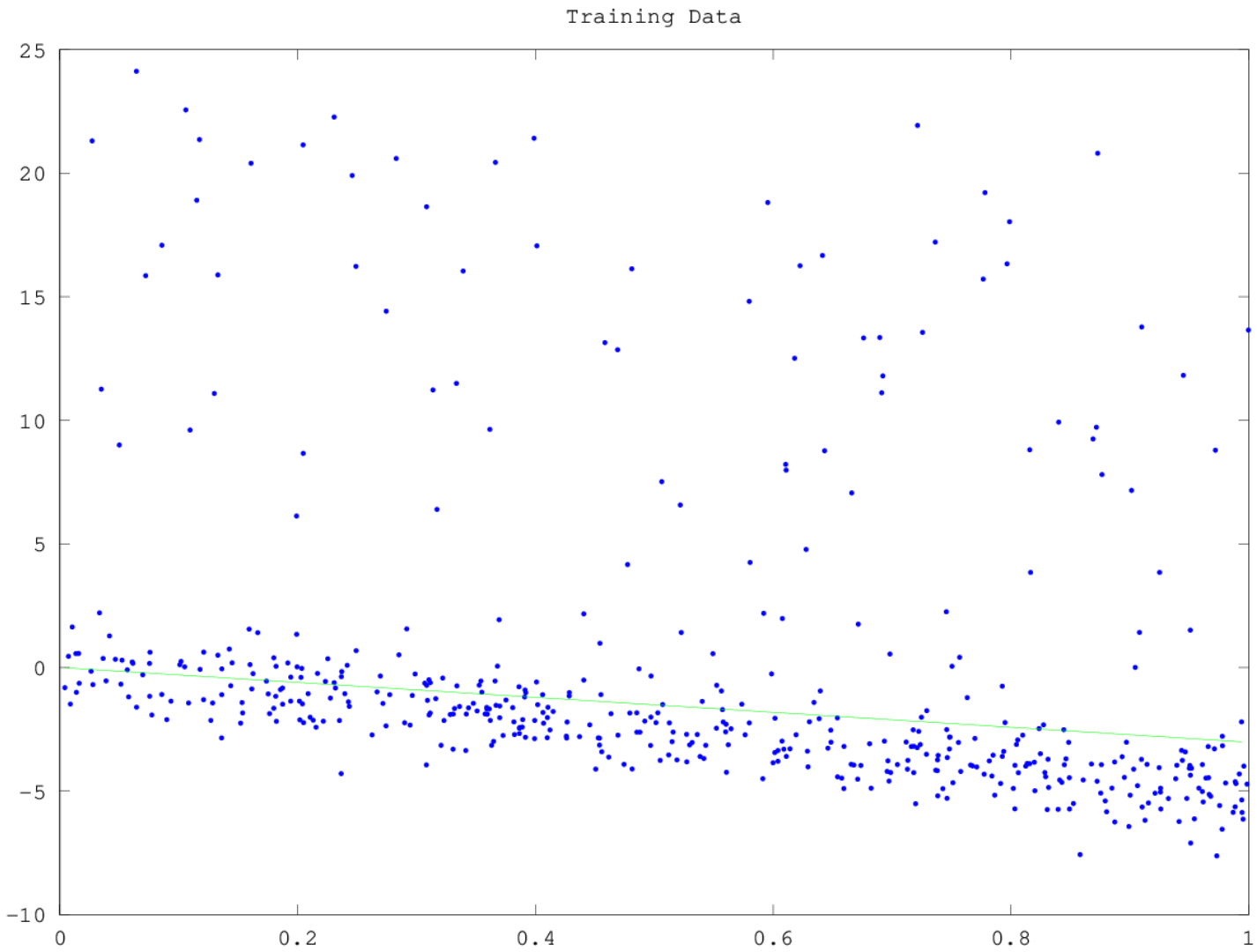
To fix these issues, you could add bias and linear basis so it acts like linear regression instead of zero when there isn't data close to that point.

4. Robust Regression and Gradient Descent

4.1. Weighted Least Squares in One Dimension

```
function [model] = weightedLeastSquares(X,y,z)
```

```
% Solve least squares problem
%w = (X'*z*X)\(X'*z*y)
w = 1/2*( (X'*z*X)\(y'*z*X) + X\y)

model.w = w;
model.predict = @predict;

end

function [yhat] = predict(model,Xhat)
w = model.w;
yhat = Xhat*w;
end
```



4.2. Smooth Approximation to the L1-Norm

$$f(w) = \sum_{i=1}^{n} \log(\exp(w^T x_i - y_i) + exp(y_i - w^T x_i))$$

$$f(w) = \sum_{i=1}^{n} \log(\exp(x_i^T w - y_i) + exp(y_i - x_i^T w))$$

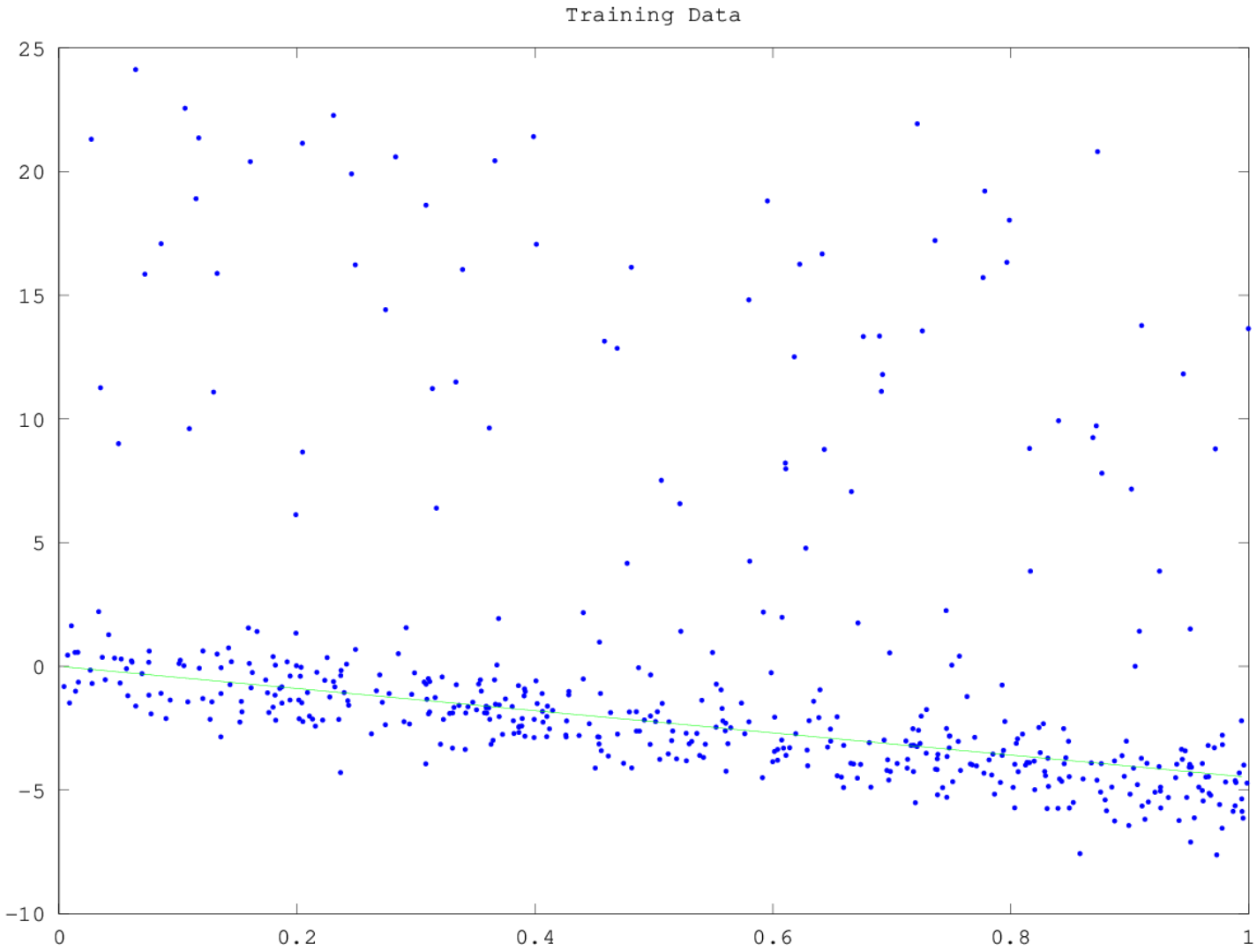
$$\frac{df(w)}{dw_j} = \sum_{i=1}^{n} \frac{\exp(x_i^T w - y_i) * x_{i,j} - exp(y_i - x_i^T w) * x_{i,j}}{\exp(x_i^T w - y_i) + exp(y_i - x_i^T w)}$$

For d features/dimensions:

$$\nabla f(w) = \begin{bmatrix} \frac{df(w)}{dw_1} \\ \vdots \\ \frac{df(w)}{dw_d} \end{bmatrix}$$

4.3. Robust Regression

```
function [f,g] = funObj(w,X,y)
  w = w';
  d = rows(w);
  n = rows(X);

  f = 0;
  for i = 1:n
      f += log(exp(w'*X(i)-y(i))+exp(y(i)-w'*X(i)));
  end
  g = zeros(d,1);
  for j = 1:d
      total = 0;
      for i = 1:n
          total += (exp(X(i)'*w-y(i))*X(i,j)-exp(y(i)-X(i)'*w)*X(i,j))/(exp(X(i)'*w-y(i))+exp(y(i)-X(i)'*w));
      end
      g(j) = total;
  end
end
```


Training Data
25
20
15
10
5
0
−5
−10
0
0.2
0.4
0.6
0.8
1