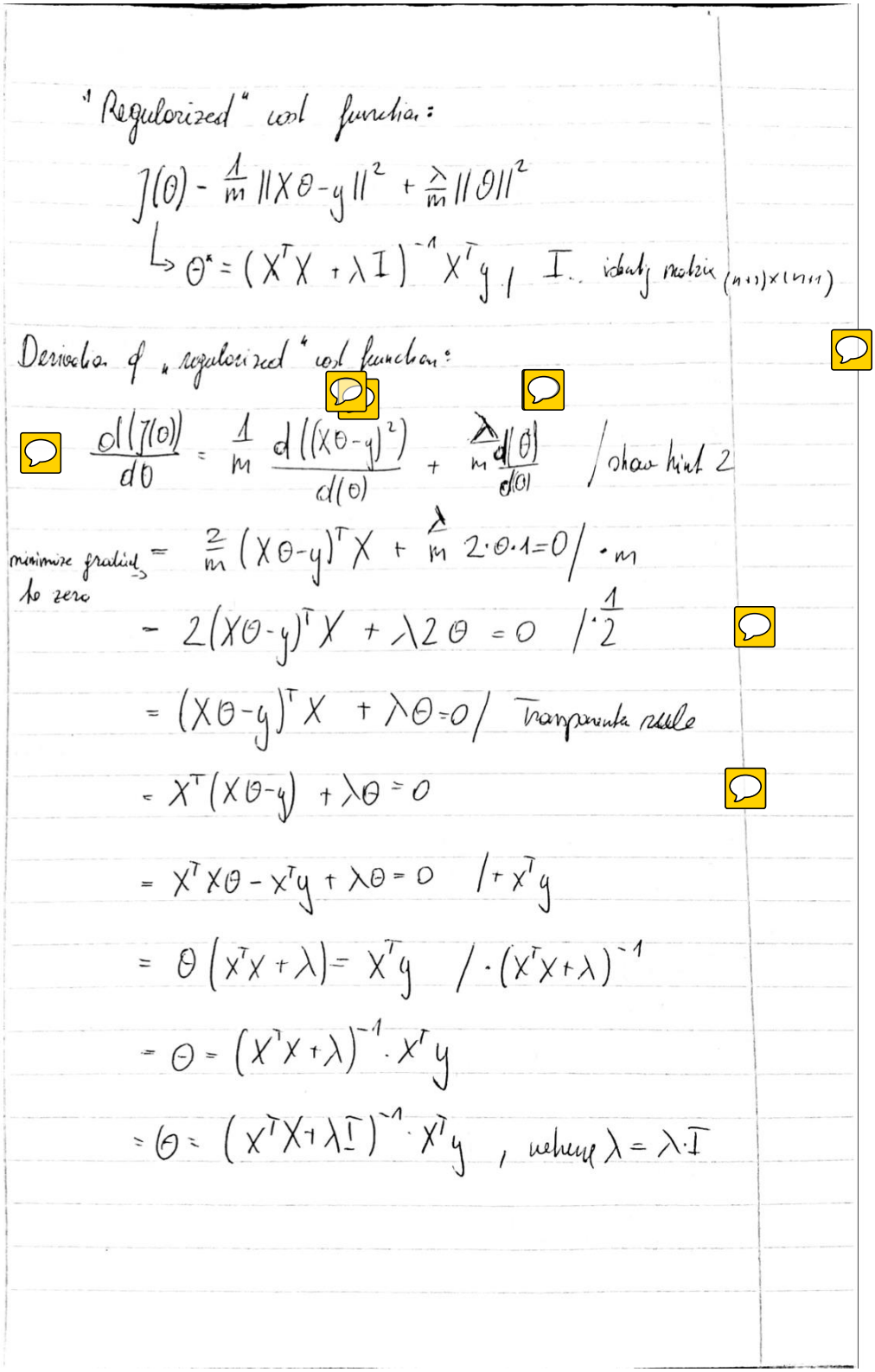
Assignment 1

Computational Intelligence SEW, SS2017

Team Members		
Last name	First name	Matriculation Number
Guggi	Simon	1430534
Papst	Stefan	1430868
Perkonigg	Michelle	1430153

Derivation of Regularized Linear Regression



"Regularized" cost function:

$$J(\theta) = \frac{1}{m}\|X\theta - y\|^2 + \frac{\lambda}{m}\|\theta\|^2$$

$$\hookrightarrow \theta^* = (X^T X + \lambda I)^{-1} X^T y \;/\; I \dots \text{identity matrix } (n+1)\times(n+1)$$

Derivation of „regularized" cost function:

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{m}\frac{\partial\left((X\theta - y)^2\right)}{\partial(\theta)} + \frac{\lambda}{m}\frac{\partial(\theta)}{\partial(\theta)} \quad / \text{chain hint 2}$$

minimize gradient to zero $\Rightarrow$

$$= \frac{2}{m}(X\theta - y)^T X + \frac{\lambda}{m} 2\cdot\theta\cdot 1 = 0 \;/\cdot m$$

$$= 2(X\theta - y)^T X + \lambda 2\theta = 0 \;/\cdot\frac{1}{2}$$

$$= (X\theta - y)^T X + \lambda\theta = 0 \;/\; \text{Transpose rule}$$

$$= X^T(X\theta - y) + \lambda\theta = 0$$

$$= X^T X\theta - X^T y + \lambda\theta = 0 \;/ + X^T y$$

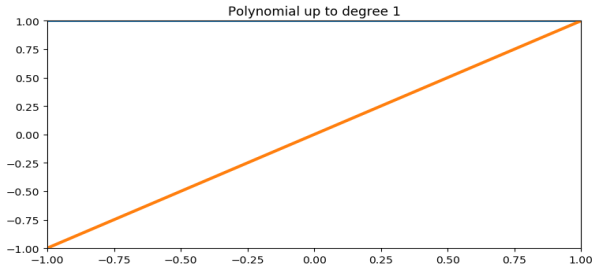
$$= \theta(X^T X + \lambda) = X^T y \;/\cdot(X^T X + \lambda)^{-1}$$

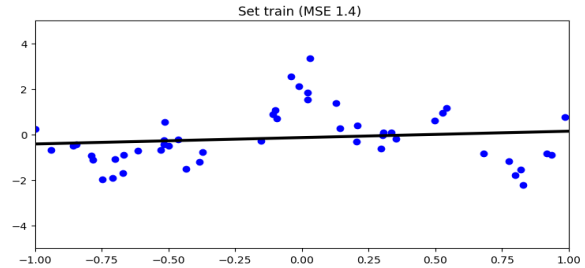
$$= \theta = (X^T X + \lambda)^{-1} \cdot X^T y$$

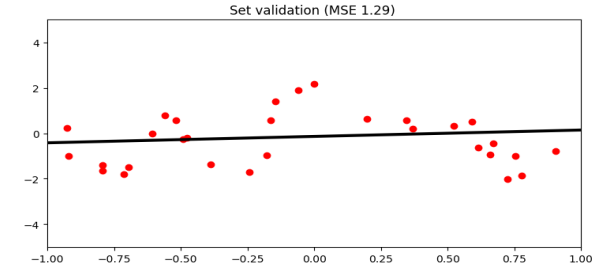
$$= \theta = (X^T X + \lambda I)^{-1} \cdot X^T y \text{, where } \lambda = \lambda \cdot I$$

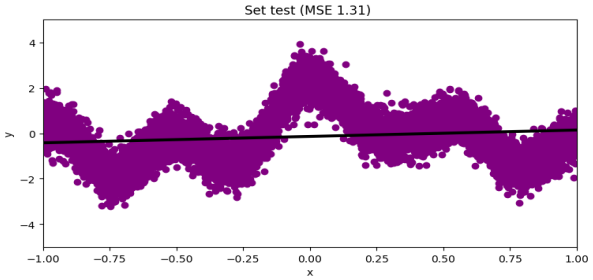
Plot for each of the following polynomial degrees: 1, 2, 5, 20

degree 1

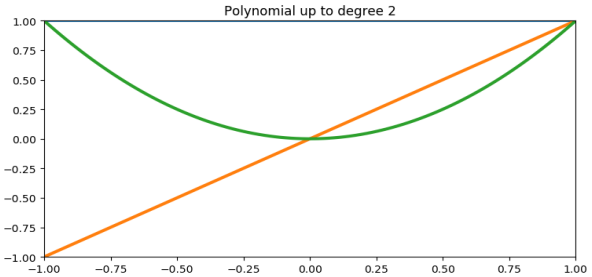


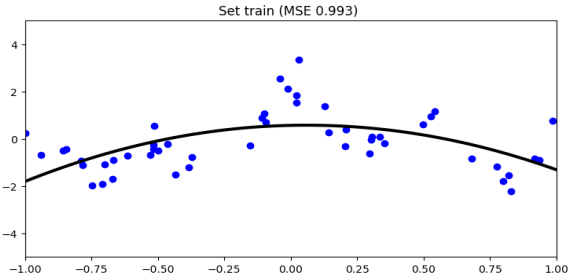


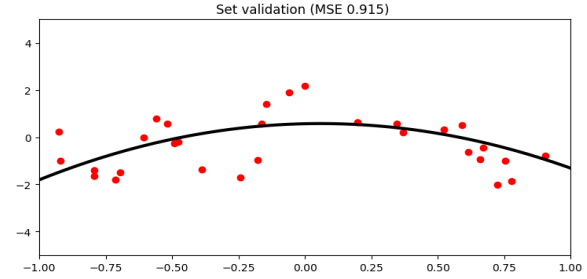


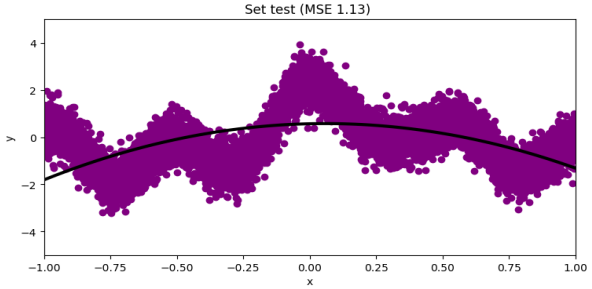


degree 2

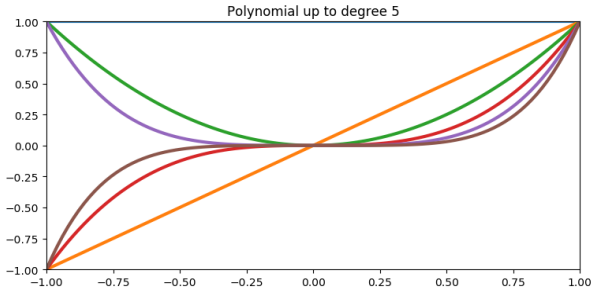


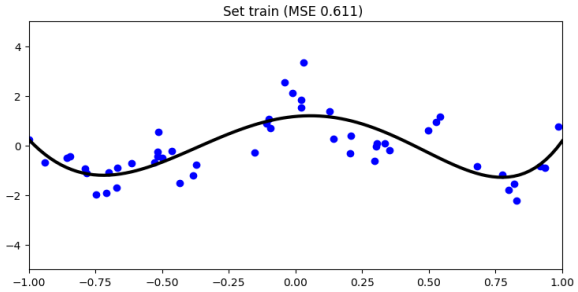


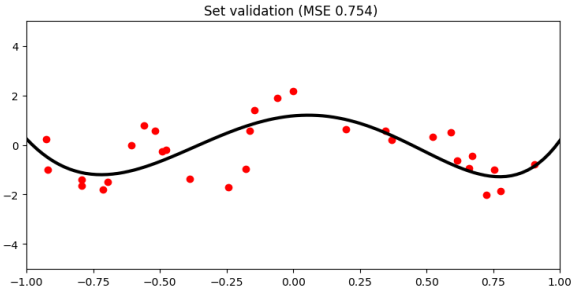


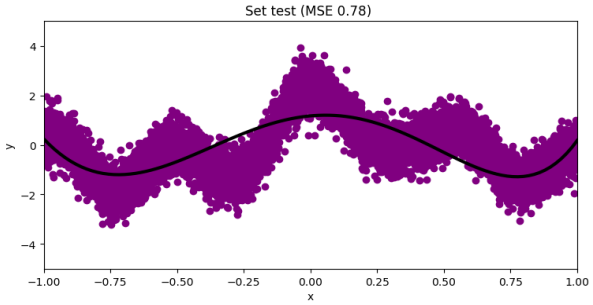


degree 5

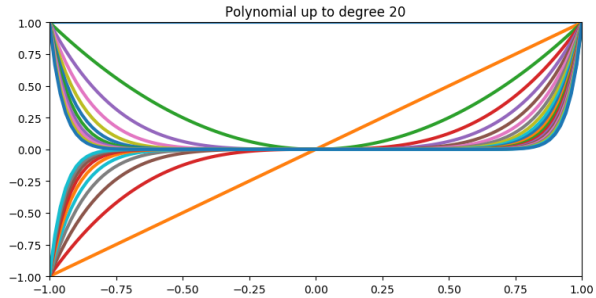


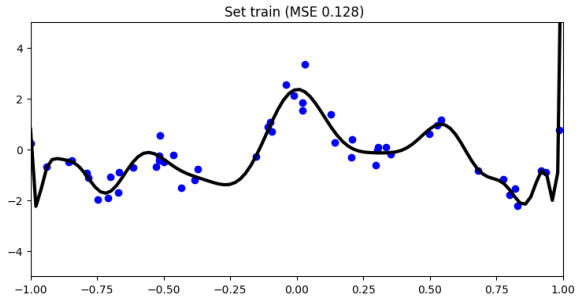


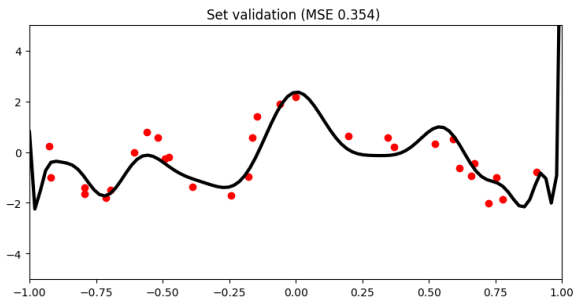


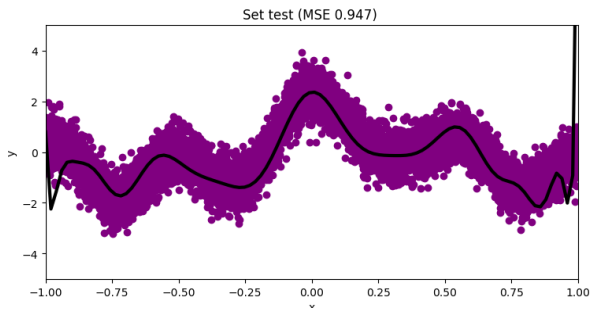


degree 20



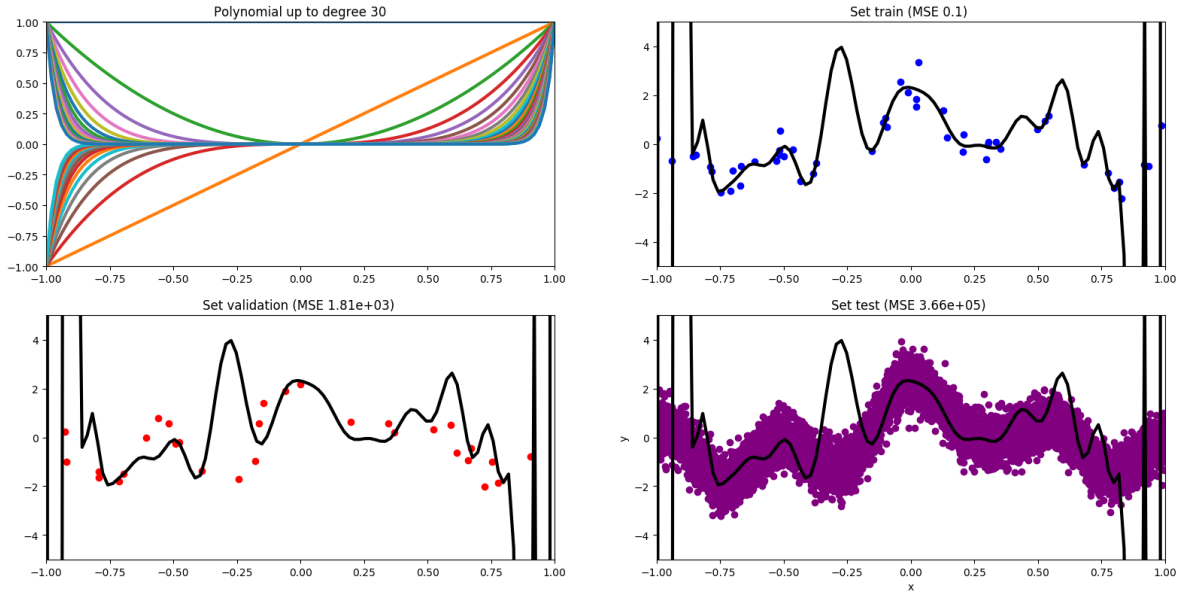






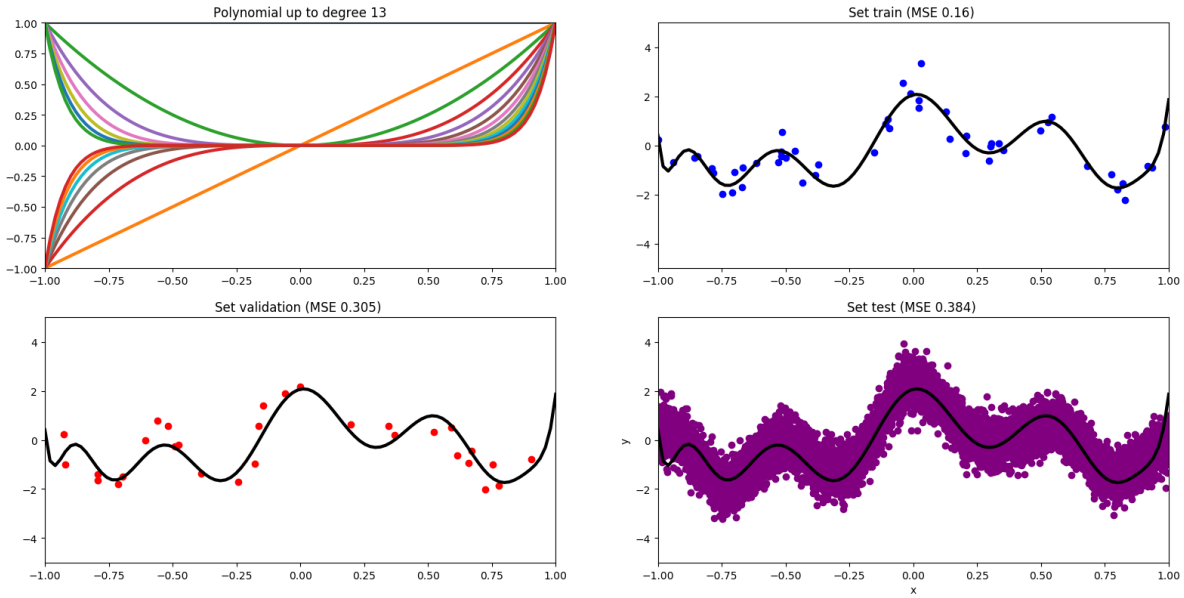
Plot of polynomial degree with lowest training error

The polynomial degree 30 has the lowest training error.
Training error: 0.10026711145129742
Test error: 366141.23594091967

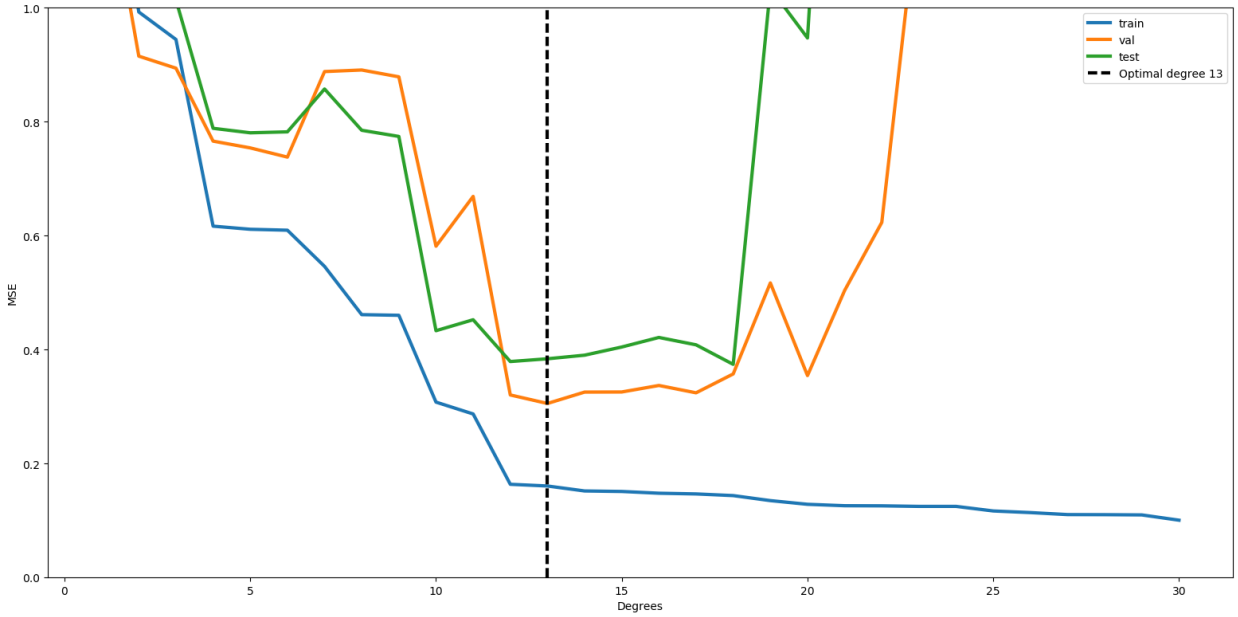


Plot of polynomial degree with lowest validation error

The polynomial degree 13 has the lowest validation error.
Validation error: 0.305228632706
Testing error: 0.3837016046508086



Plot of training, validation and testing errors as a function



Discuss your findings in your own words using the concept of over-fitting. Why is it important to use a validation set?

Overfitting: When a model is overfitted it is too complex, because of too many parameters. Overfitting leads to a low training error but a very high testing error.

	polynomial degree 13	**polynomial degree 30**
training error	0.1603029754981785	0.1002671114512974
validation error	0.3052286327055114	1805.9699422675815
testing error	0.3837016046508086	366141.23594091967

The table shows a comparison between a model with the lowest training error and a model with the lowest validation error. As you can see the results differ significantly. While the training error of the polynomial degree 30 is very small, the validation and testing errors are extremely high. This type of model is called as overfitted. The polynomial degree 13 shows a moderate training error, but a very small validation error. The validation set is used to avoid overfitting. The right fitted model is the one with the smallest testing error.

The plot of the training, validation and testing errors as a function shows that the blue line, the training errors have its lowest point at the polynomial degree 30. It seems like the training error function is always decreasing. The orange line, the validation error has its lowest point at the polynomial degree 13 and the green line, the testing errors have its lowest point at the polynomial degree 18.

Derivation of Gradient

Derivation of gradient:

$h_\theta(x) = \sigma(x^T\theta) \qquad \sigma(z) = \frac{1}{1+e^{-z}}$

$\sigma(z)' = \sigma(z)\cdot(1-\sigma(z))$

$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(h_\theta(x^{(i)})\right) + (1-y^{(i)})\cdot\log\left(1-h_\theta(x^{(i)})\right)\right)$

for one example θ_i:

$J(\theta_i) = -\frac{1}{m}\left(y\cdot\log\left(h_\theta(x)\right) + (1-y)\cdot\log\left(1-h_\theta(x)\right)\right)$

$= -\frac{1}{m}\left(y\cdot\log\left(\sigma(x^T\theta)\right) + (1-y)\cdot\log\left(1-\sigma(x^T\theta)\right)\right)$

$\frac{d J(\theta_i)}{d\theta} = -\frac{1}{m}\left(y\cdot\frac{1}{\sigma(x^T\theta)}\cdot\sigma(x^T\theta)\cdot(1-\sigma(x^T\theta))\cdot x + (1-y)\cdot\frac{1}{(1-\sigma(x^T\theta))}\cdot(1-\sigma(x^T\theta))\cdot\underbrace{(1-(1-\sigma(x^T\theta)))}_{=-\sigma(x^T\theta)}\cdot x\right)$

$= -\frac{1}{m}\left(y\cdot(1-\sigma(x^T\theta))\cdot x + (1-y)\cdot -\sigma(x^T\theta)\cdot x\right)$

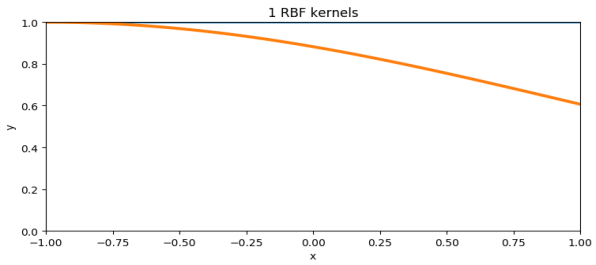
$= -\frac{1}{m}\left(y\cdot(1-h_\theta(x))x + (1-y)\cdot -h_\theta(x)\cdot x\right)$

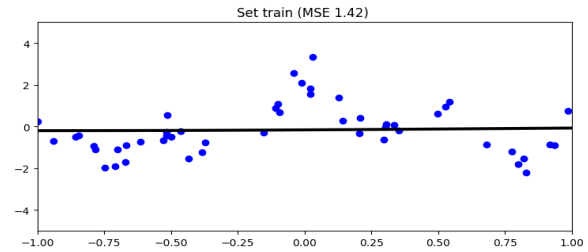
$= -\frac{1}{m}\left(y\cdot(1-h_\theta(x)) + (1-y)\cdot -h_\theta(x)\right)\cdot x$

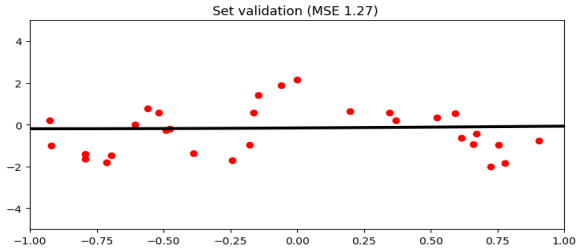
$= -\frac{1}{m}\left(y - h_\theta(x)\cdot y - h_\theta(x) + h_\theta(x)\cdot y\right)\cdot x = -\frac{1}{m}\left(y - h_\theta(x)\right)\cdot x$

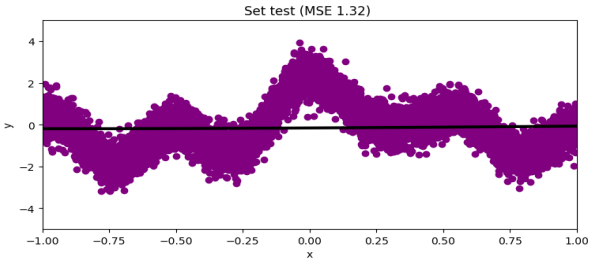
Plot for each of the following degrees: 1, 2, 5, 20

degree 1

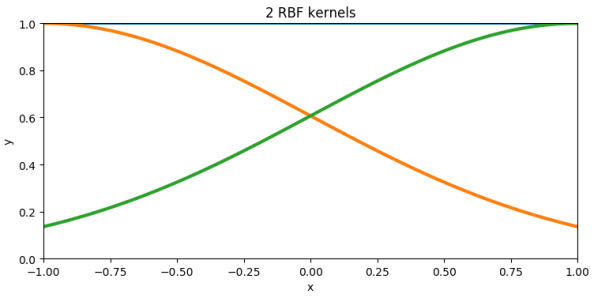


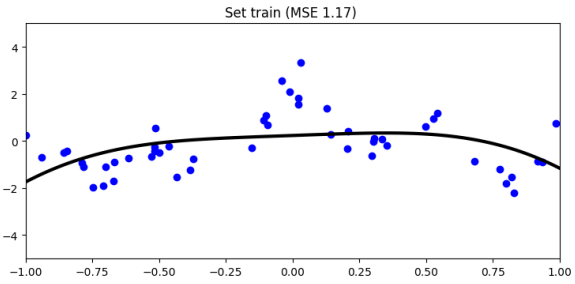


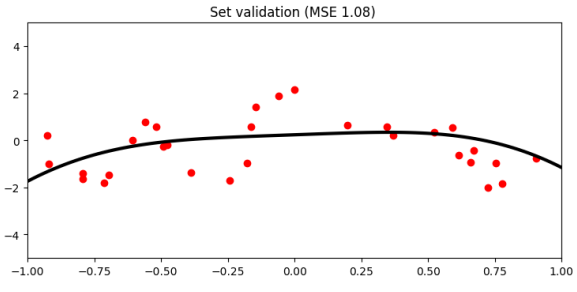


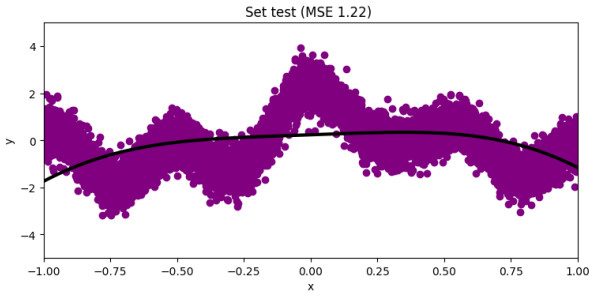


degree 2

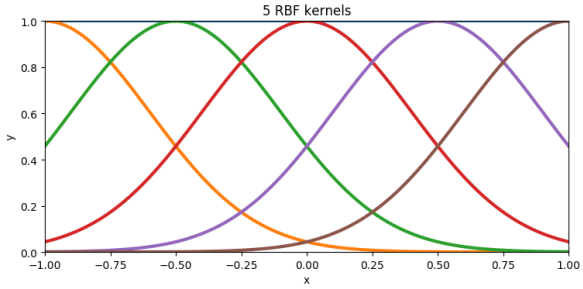


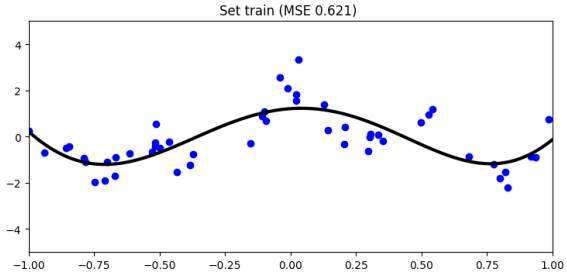


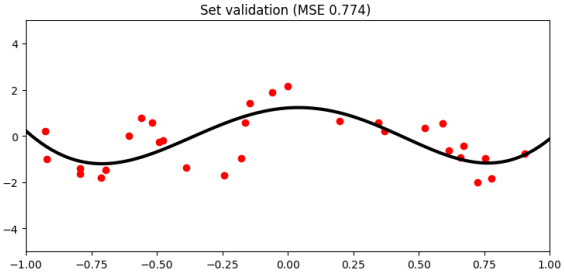


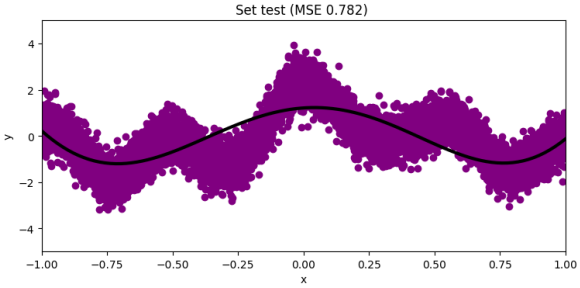


degree 5

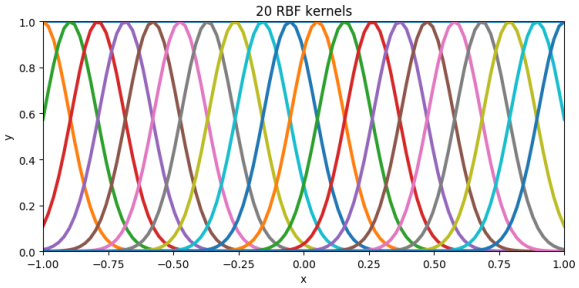


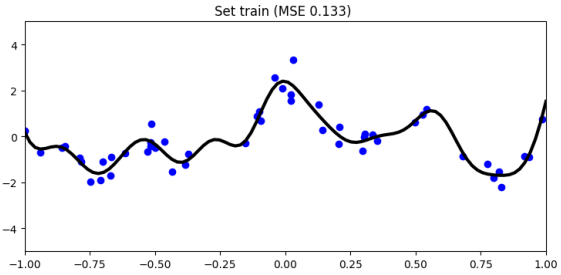


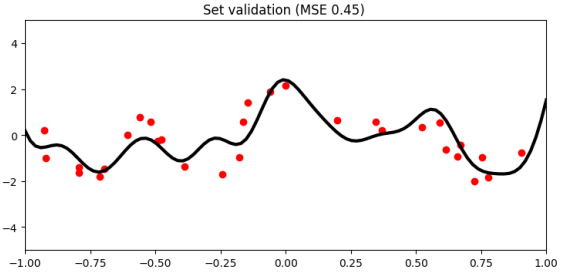


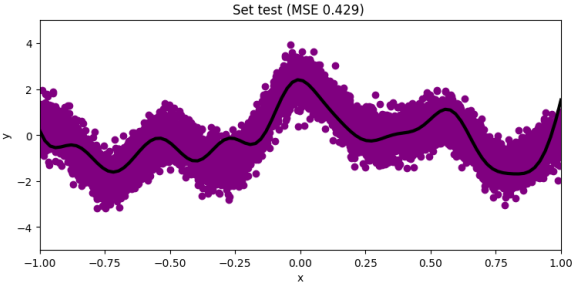


degree 20



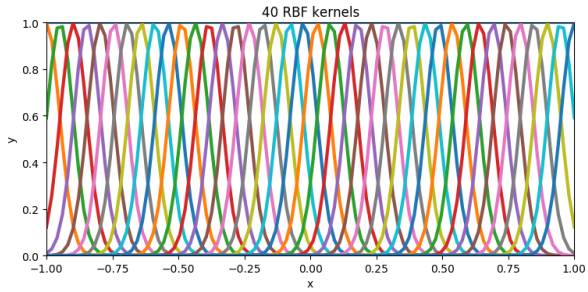


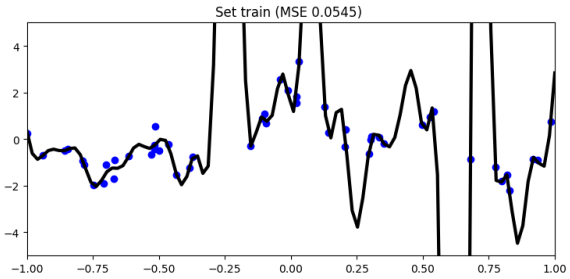


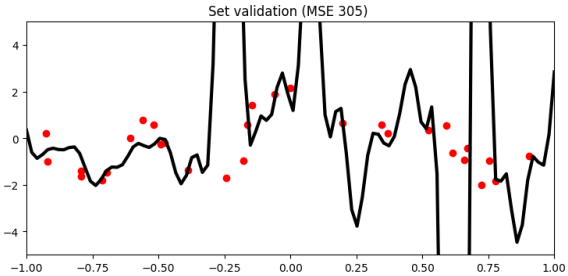


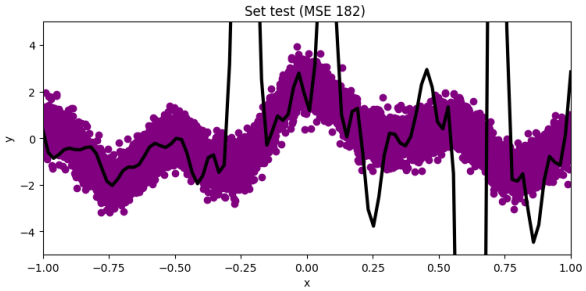
Plot of RBF with lowest training error

The RBF 40 has the lowest training error.
Training error: 0.054452916301051674
Testing error: 181.67182614369543



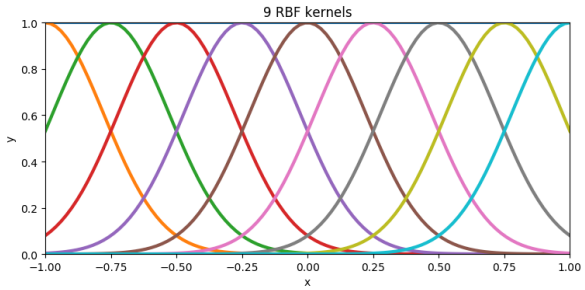


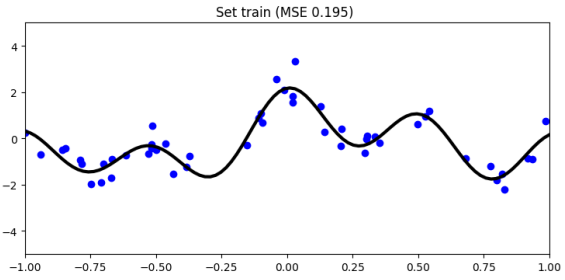


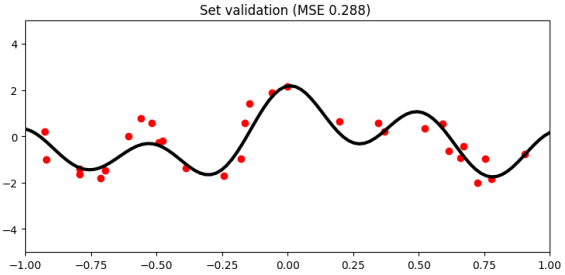


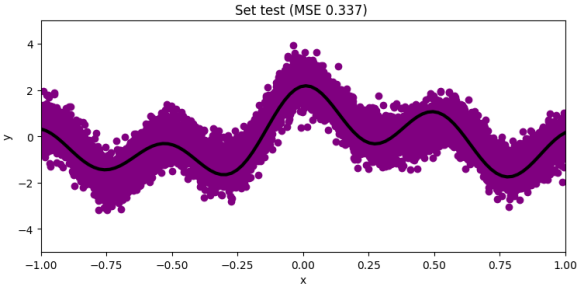
Plot of RBF with lowest validation error

The RBF 9 has the lowest validation error.
Validation error: 0.2879598418721303
Testing error: 0.3370130082408977

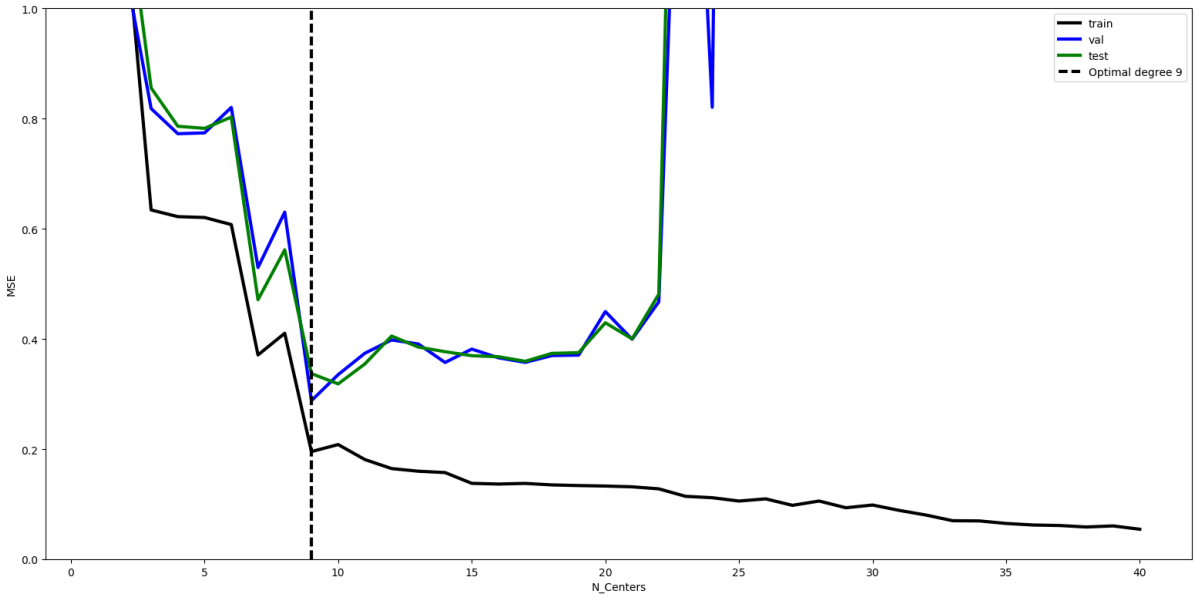








Plot of training, validation and testing errors as a function



Briefly describe and discuss your findings in your own words.

The RBF with the lowest training error is, as with the polynomial model, the highest degree tested (degree 40). The testing error is again very high and therefore the RBF is overfitted. The lowest validation error achieved the RBF with degree 9. This one has a significant lower testing error than the RBF with degree 40. In this case the RBF model with the lowest validation error has the same degree as the RBF model with the lowest testing error. Both have degree 9, as you can see in the plot of training, validation and testing errors as a function. The black line represents the training errors, the blue line the validation errors and the green line the testing errors.

Is the polynomial or the RBF model better?

The RBF model is better.

The function check_gradient in toolbox.py is here to test if your gradient is well computed. Explain what it is doing.

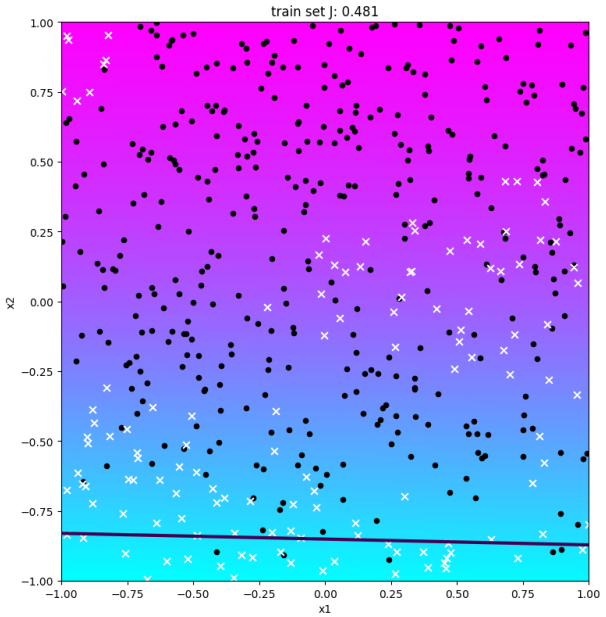
The function checks if the calculated gradients are well computed by finite difference approximation. The finite difference approximation is calculated on the following basis:

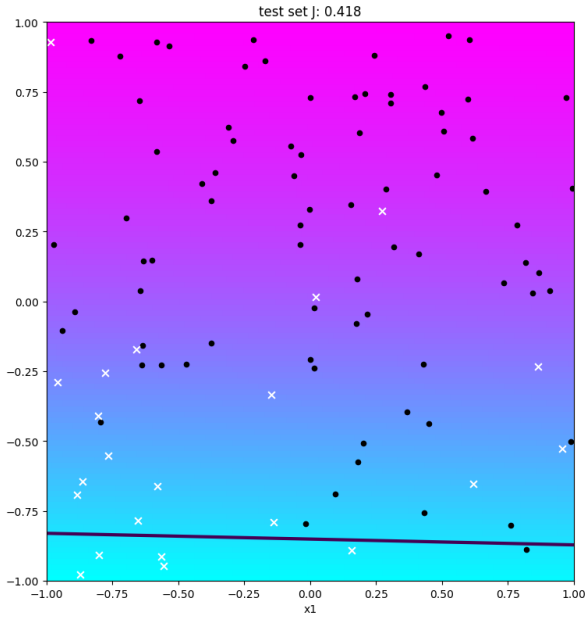
$$\frac{\big(f(x + eta * dx) - f(x)\big)}{eta} \; converges\; toward\; grad * dx$$

It takes the cost – function, the gradient – function and the number of features as parameters. 'tries' and 'deltas' have default values. For each try the function calculates the differences quotient of a random number and a random index of the randomly calculated vector. If each diff value is smaller than -1 or each value in approx._err is smaller than -20, the check passes.

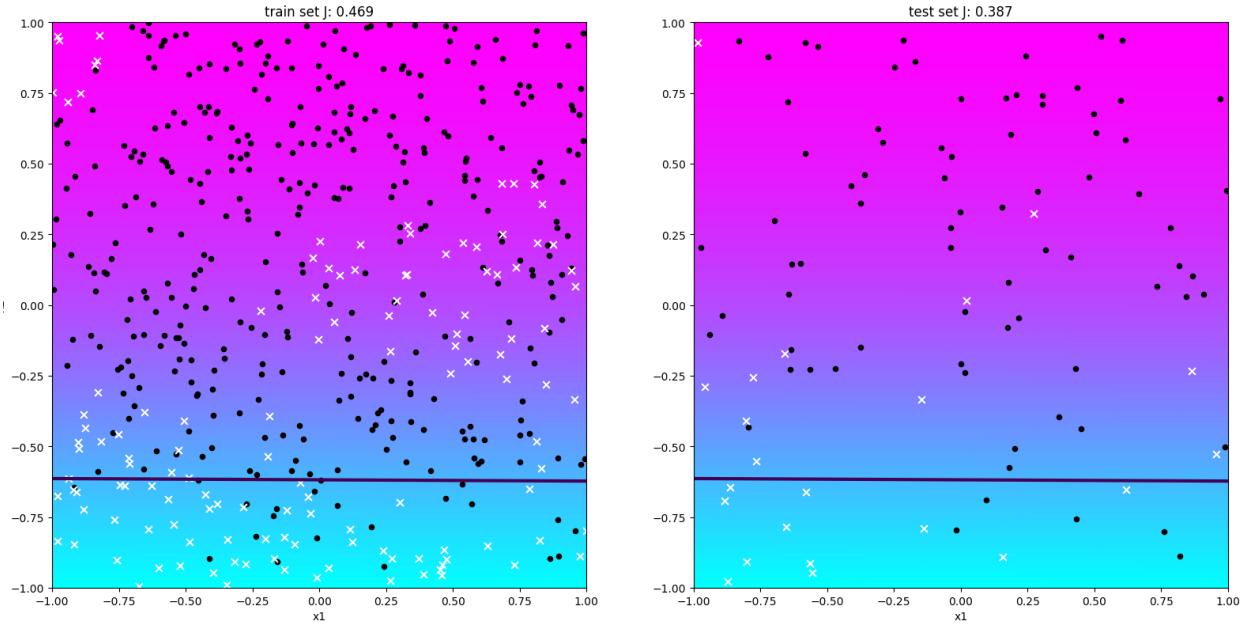
For degree l = 1 run GD for 20 and 2000 iterations (learning rate η = 1, all three parameters initialized at zero). Report training and test errors for each iteration number and plot the decision boundaries. Comment on the results and explain why the number of iterations should be neither too low nor too high.

degree: 1
eta: 1
iterations: 20
training error: 0.481
testing error: 0.418





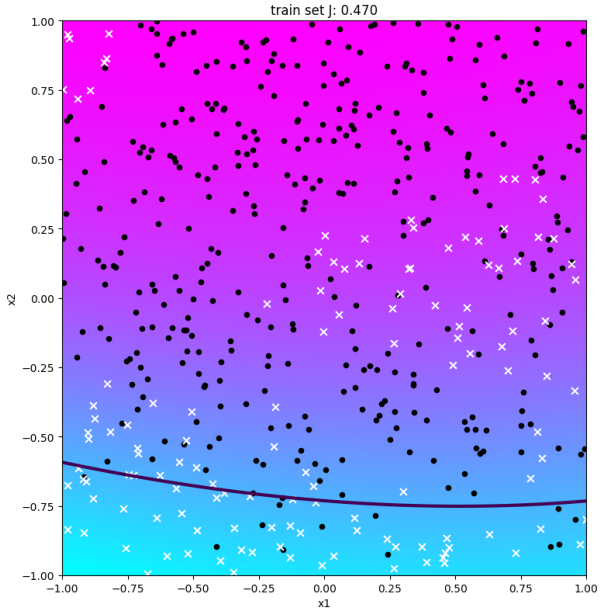
degree: 1
eta: 1
iterations: 2000
training error: 0.469
testing error: 0.387

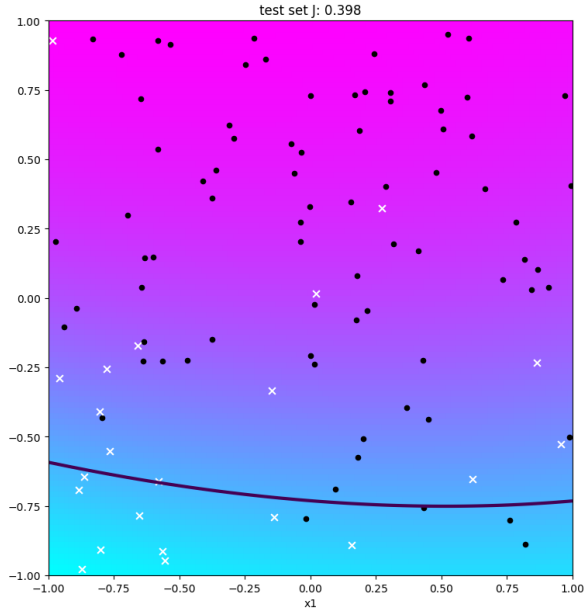


Regarding the number of iterations there are two keywords important: overfitting and underfitting. If there are too less iterations this leads to underfitting, where the model is too simple. On the other side if there are too much iterations this leads to overfitting, where the model is too complex.

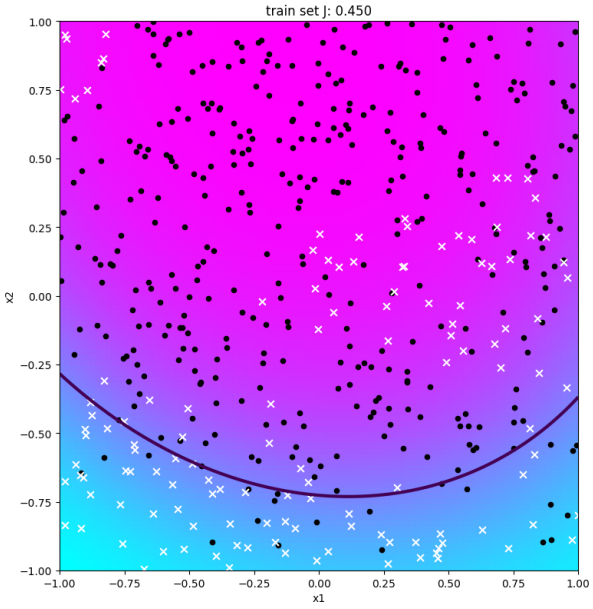
For degree l = 2 run GD for 200 iterations and learning rates of η = .15, η = 1.5 and η = 15.. Report training and test errors for each iteration number and plot the decision boundaries. Comment on the results and explain what is happening when η is too large or too small.

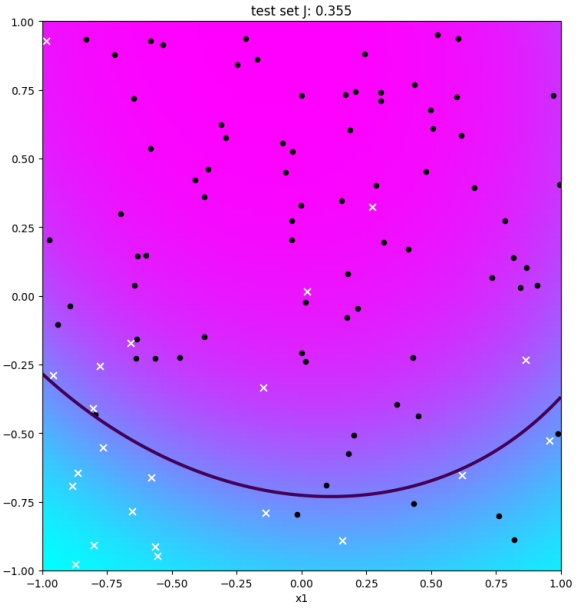
degree = 2
eta = .15
iterations = 200
training error: 0.470
testing error: 0.398



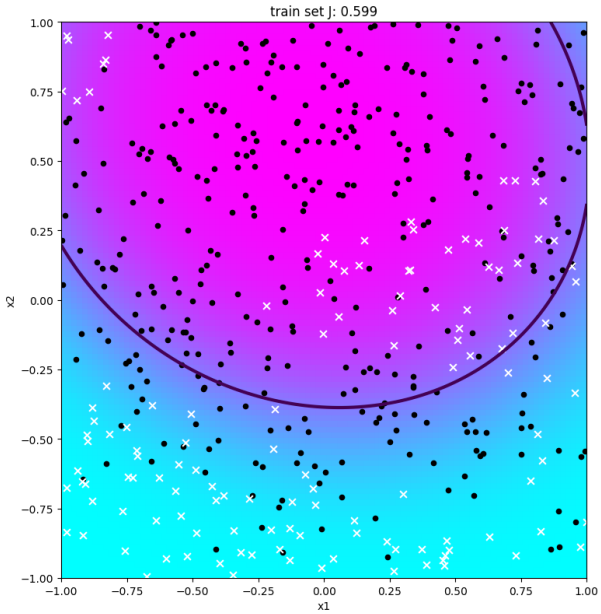


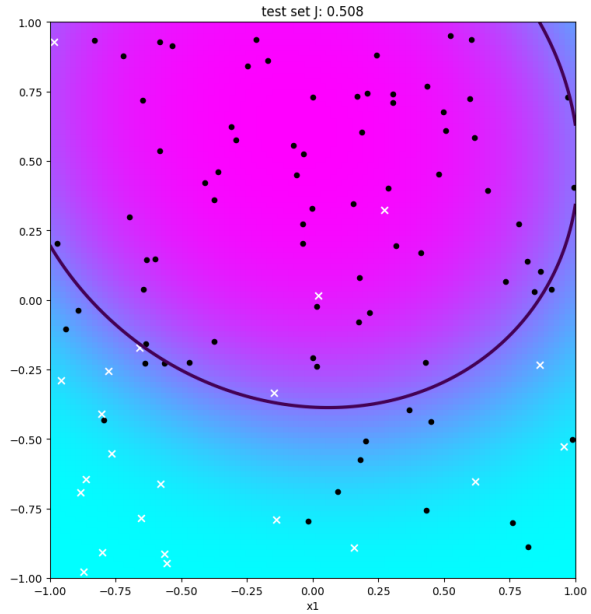
degree = 2
eta = 1.5
iterations = 200
training error: 0.450
testing error: 0.355





degree = 2
eta = 15.
iterations = 200
training error: 0.599
testing error: 0.508

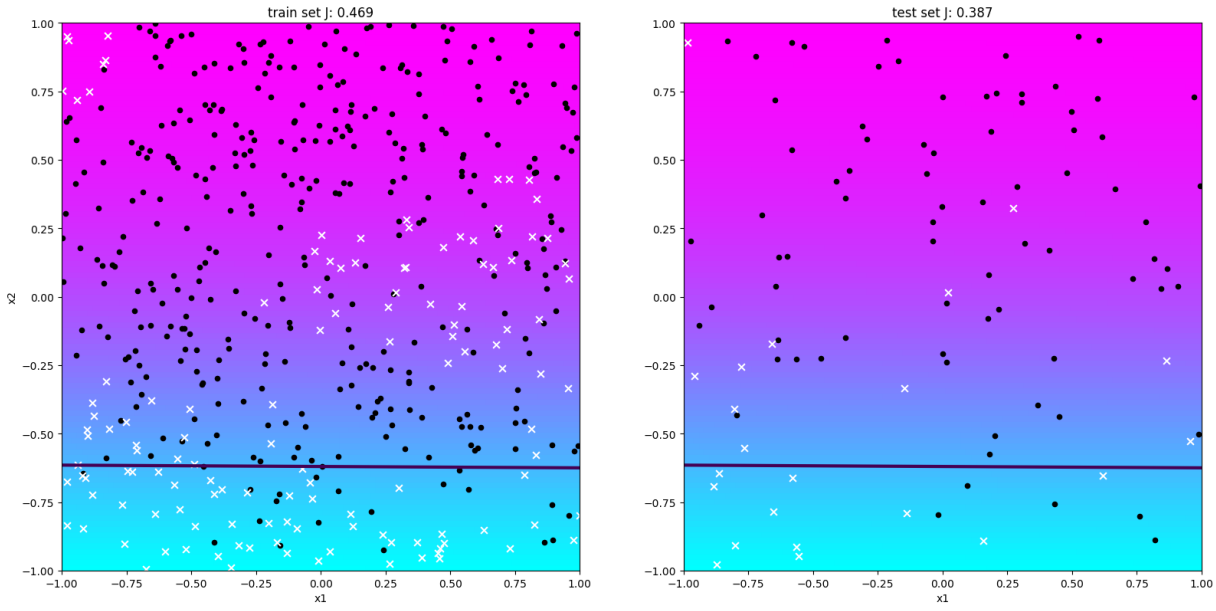




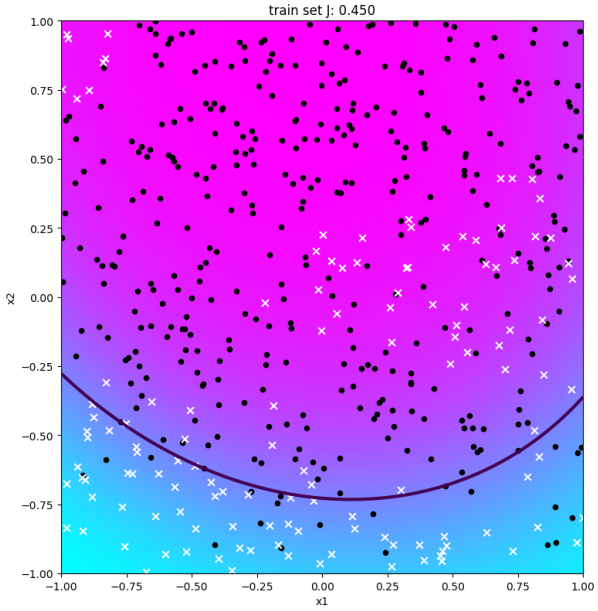
If the learning rate is too small it leads to a slow convergence, otherwise if the learning rate is too leads to oscillations, divergence.

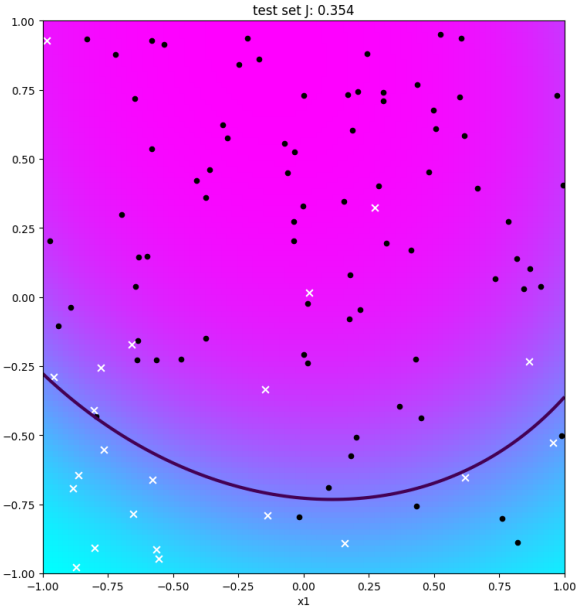
Identify reasonably good pair of values for the number of iterations and learning rate for each degree in l ϵ {1, 2, 5, 15} that provides a good solution in reasonable time. Report these pairs, the performance obtained on training and testing set and plot each solution. Conclude by describing which degree seems the most appropriate to fit the given data.

degree: 1 → iterations: 90; learning rate = 1.5; training error: 0.469; testing error: 0.387

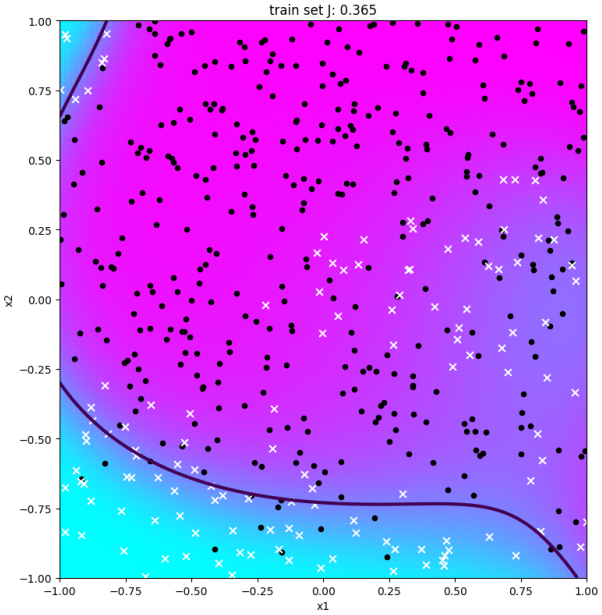


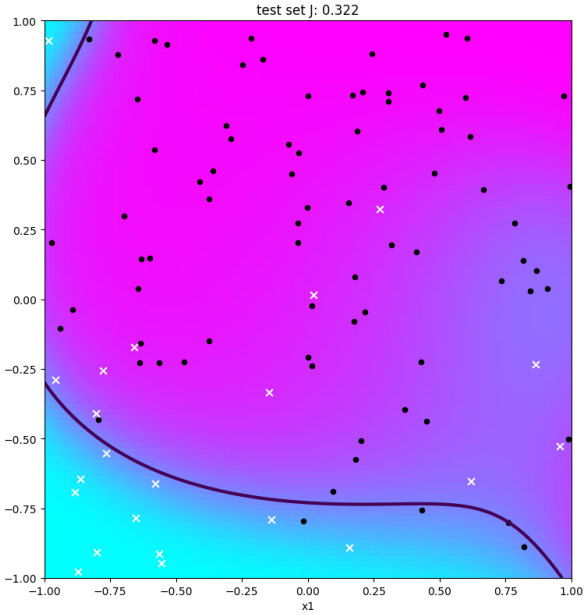
degree: 2 → iterations: 250; learning rate = 1.5; training error: 0.450; testing error: 0.354



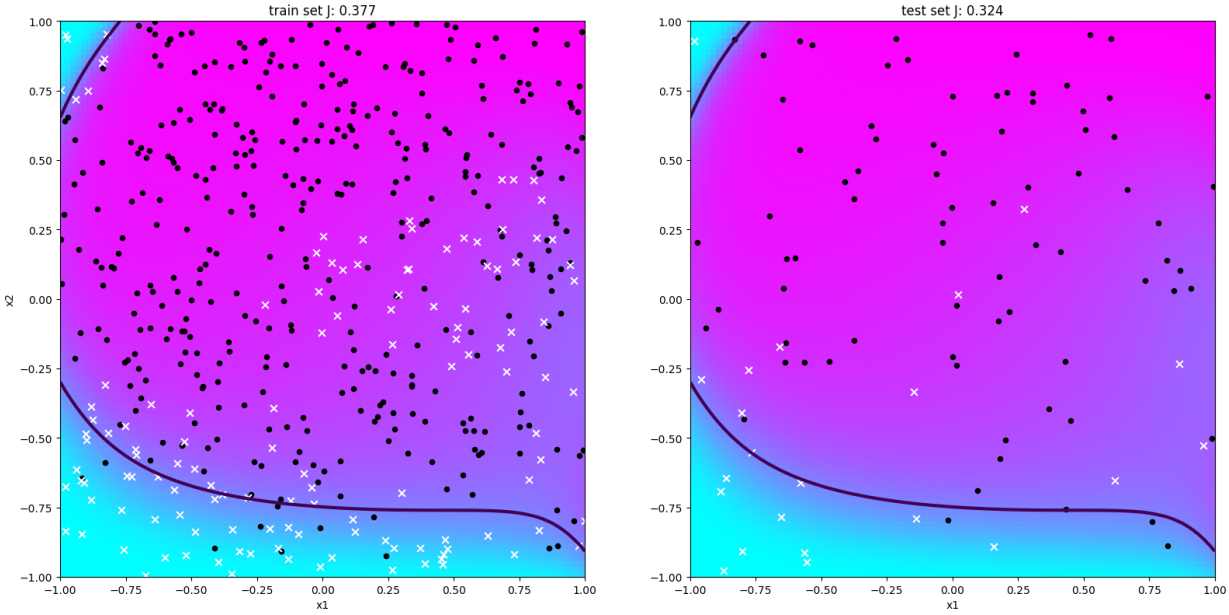


degree: 5 → iterations:290; learning rate = 1.5; training error: 0.365; testing error: 0.322





degree: 15 → iterations:80; learning rate = 1.5; training error: 0.377; testing error: 0.324

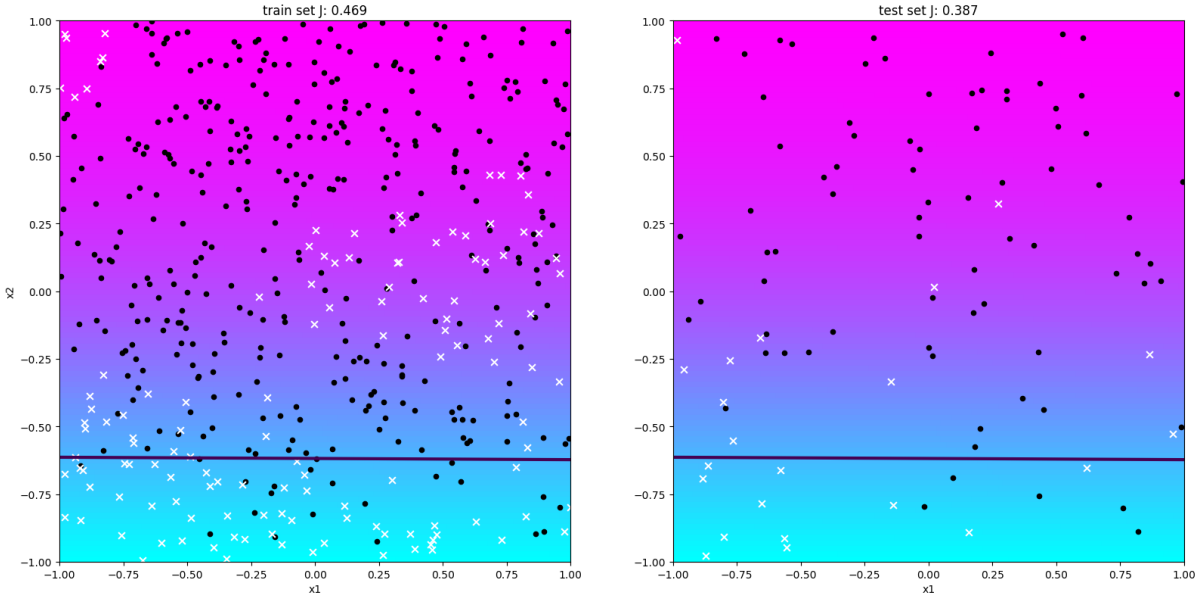


Stopping criterium to avoid doing too many iterations

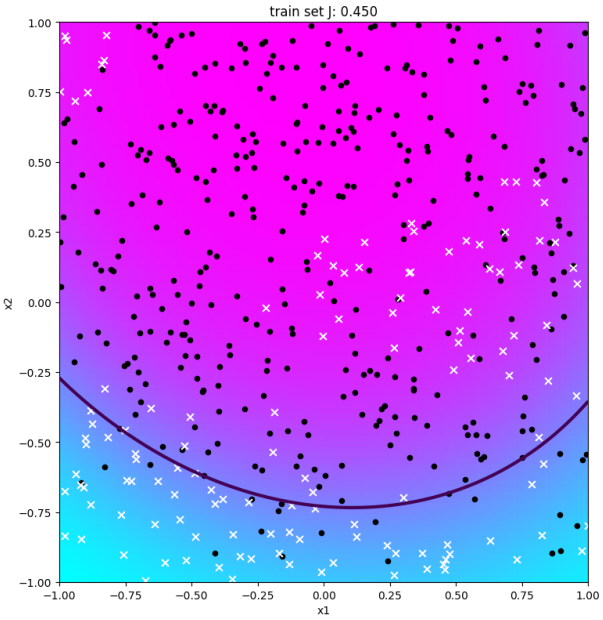
Early Stopping is a method to avoid overfitting. After every iteration, the performance is evaluated on the testing set. Up to a certain point the performance of the training set is improved but after this point the performance decreases.

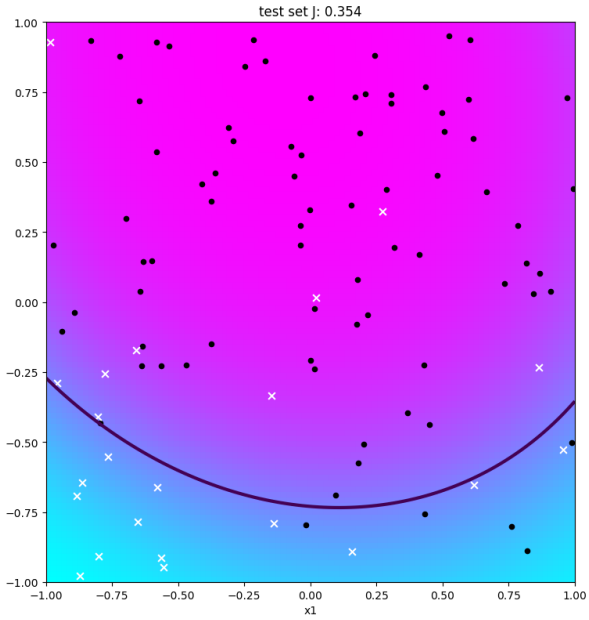
Run GDad for varying degrees l ϵ {1, 2, 5, 15} (with zero initialization of parameters, 1000 iterations, and initial learning rate η = 1). Report training and test errors, final learning rates and plot the hypothesis obtained in each case with function plot_logreg in file logreg_toolbox.py.

degree: 1
final learning rate: 0.385
training error: 0.469
testing error: 0.387

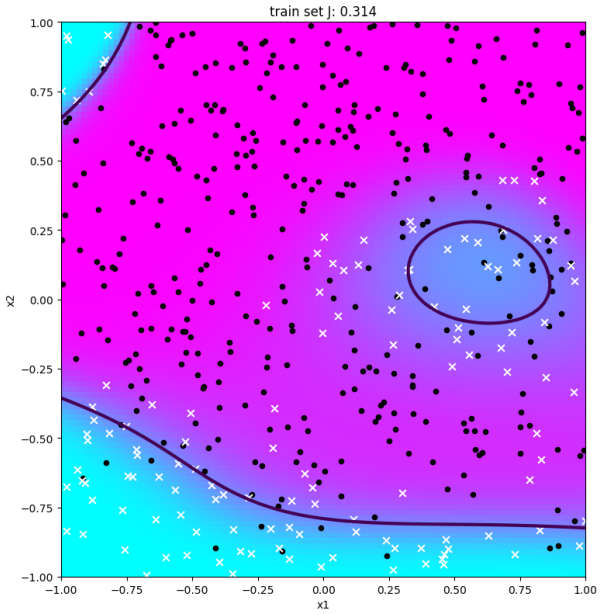


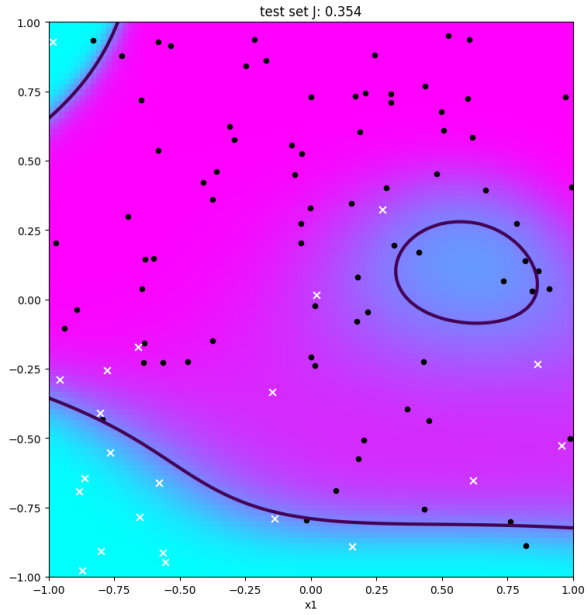
degree: 2
final learning rate: 5.11e-08
training error: 0.450
testing error: 0.354



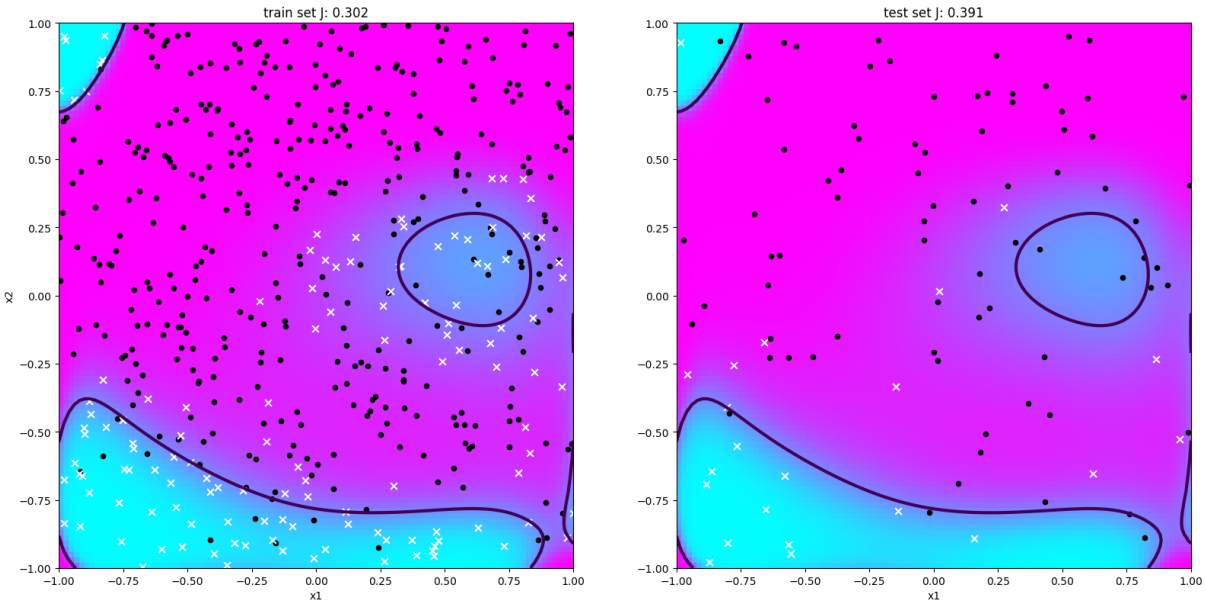


degree: 5
final learning rate: 12.5
training error: 0.314
testing error: 0.354





degree: 15
final learning rate: 12.5
training error: 0.302
testing error: 0.391



For each degree compare with the non-adaptive gradient descent variant. In particular, is the final learning rate numerically coherent with your previous guess?

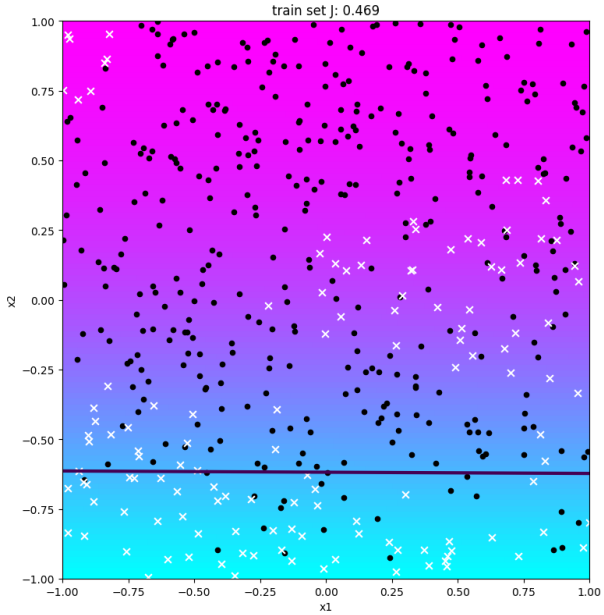
The adaptive final learning rate is numerically coherent, because it is 1,81 times the learning rate at the beginning.

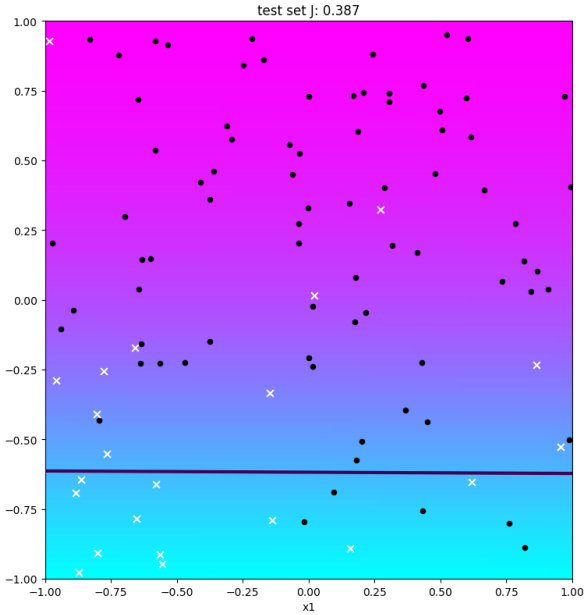
Discuss why GDad can be useful.

GDad is useful, because it often eliminates slow convergence and divergence issues.

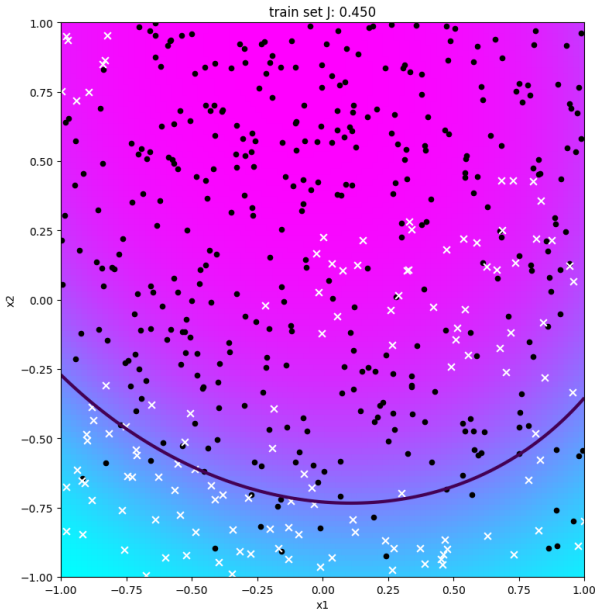
Report training and test errors, and plot the hypothesis obtained for the four degrees l ϵ {1, 2, 5, 15} with function plot_logreg.

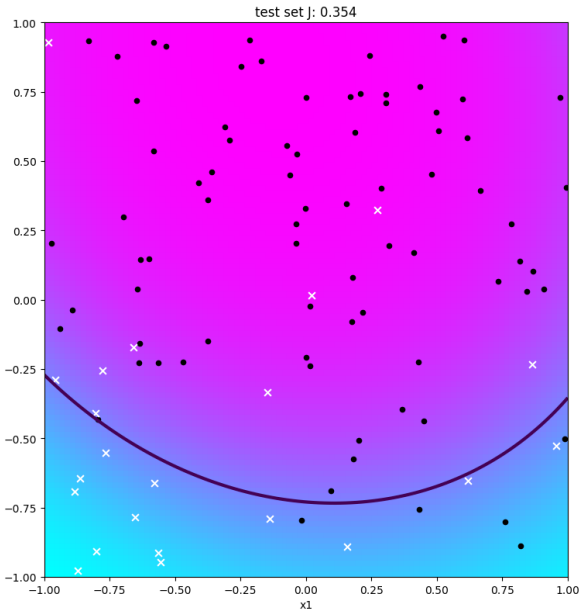
degree: 1
training error: 0.469
testing error: 0.387



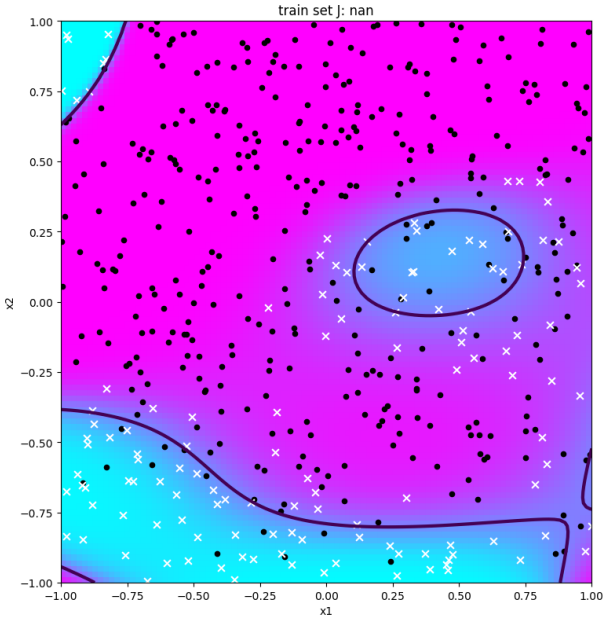


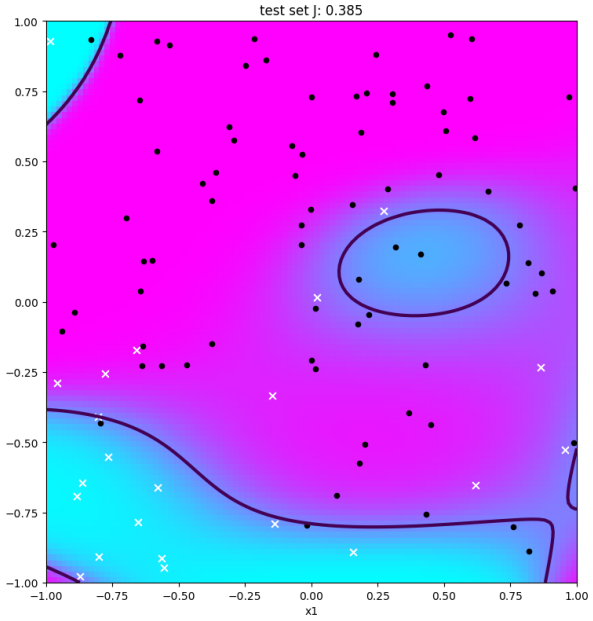
degree: 2
training error: 0.450
testing error: 0.354



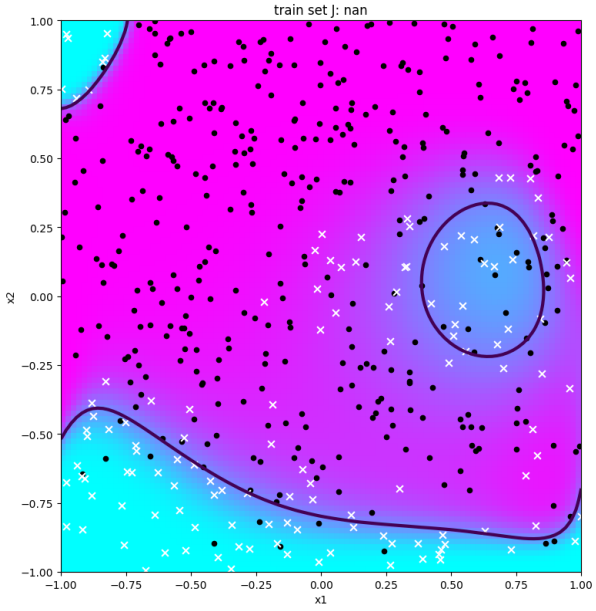


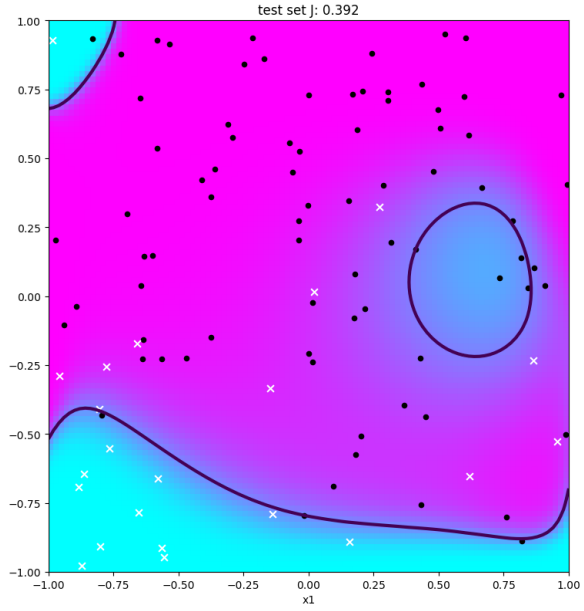
degree: 5
training error: nan
testing error: 0.385





degree: 15
training error: nan
testing error: 0.392





Compare the results with those above (GD/GDad) and briefly discuss and interpret your observations. In particular does it change your opinion on which degree is best to fit the data?

GD → degree 5 is best to fit the data
GDad → degree 5 is best to fit the data
optimizer → degree 2 is best to fit the data