Least-Squared Error FIT

```
Find the linear combination of basis functions which best model the data.

Inputs:

x - Vector with observation locations in 1D. (indep. variable)
t - Vector with observations in 1D. (dep. variable)
params - Parameters for the basis functions to be used in func, e.g. as
  produced by gauss_basis.
func - Function handle which evaluates a basis function with parameters
  given by the columns of params and at the specified locations. e.g.
  @gauss_basis, or @hat_basis.
  For example, the first basis function at x = 2 is func(2, params(:,1)).
mu - Scalar representing the standard deviation of the prior Gaussian on
  the model parameters.

Outputs:
w - Coefficients used to generate a linear combination of the basis
  functions which is the maximum likelihood learned model.
```

```
function [w] = lsefit(x, t, params, func, mu)
    %construct design matrix A using x and func (hat/gassian basis)
    A = eval_basis(params, func, x);

    %Write down LHS and RHS of normal equation X, b
    X = A'*A + (1/mu^2)*eye(size(params,2));
    b = A'*t;

    %solve Xw = b for regularized MLE w via conjugate gradient
    w = linsolve(X,b);
end


%helper functions (tiny modifications of func_hat and func_hat) used in lab
%to help plot out the data easily in a continuous manner.
%it takes in single scalar datapoint x and M parameters of the hat/gauss
%basis function phi_j and returns vector [phi_1(x),...,phi_M(x)].

%we dot this with learned parameters w to make predictions for t given x.
function [v] = point_hat(x, params)
    c = 0.5*(params(1,:) + params(2,:));
    v = (x < c) .* (x - params(1,:))./(c - params(1,:));
    v = v + (x >= c) .* (1 - (x - c)./(params(2,:) - c));
    v(x < params(1,:)) = 0;
    v(x > params(2,:)) = 0;
end

function [v] = point_gauss(x, params)
    v = (1/(params(2)*sqrt(2*pi))) .* exp(-(x - params(1,:)).^2 ./ (2*params(2)^2));
end
```

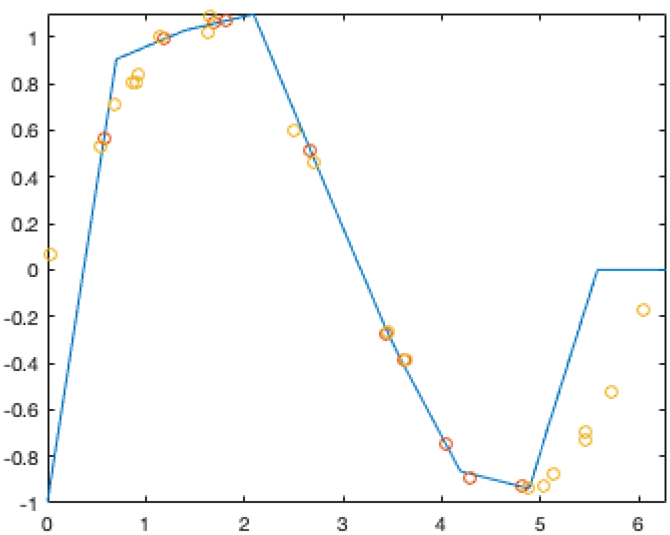
Contents

3.2.1 preparation, declare choice of basis (hat)

```
load simple.mat
load test.mat
B = hat_basis(0,2*pi,10);
```

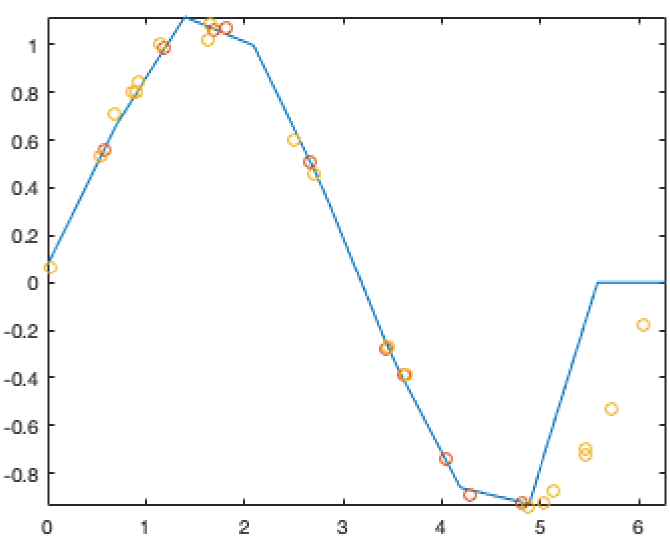
3.2.2 hat basis linear regression w/ #basis = 10, mu = 10^5

```
%learn predictor parameters
w = lsefit(x, t, B, @func_hat, 10^5);

%use learned parameters w to predict t given x, by doing dot product
%with (phi_0(x),...,phi_m(x))
f = @(s) point_hat(s, B)*w;

figure;
set(gcf,'position',[10,10,400,300])
fplot(f,[0, 2*pi]); hold on;
scatter(x,t); hold on;
scatter(test_x,test_t);
%red - training data (x,t), yellow - test data (test_x,test_t)
```



3.3.1 same as previous, but with mu = 10

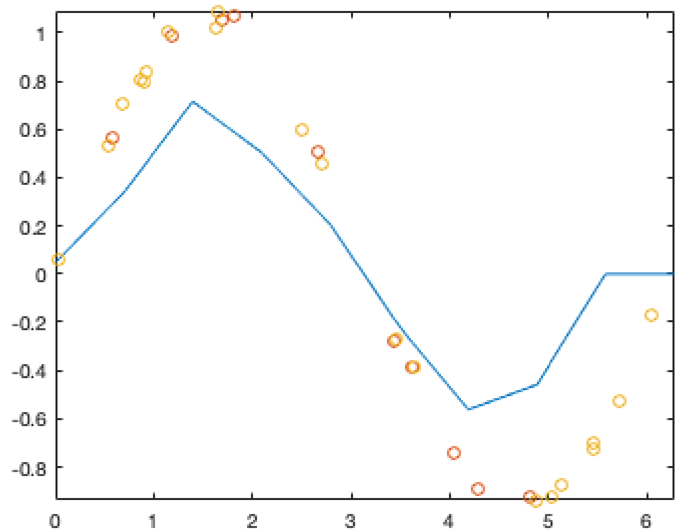
```
w = lsefit(x, t, B, @func_hat, 10);
f = @(s) point_hat(s, B)*w;

figure;
set(gcf,'position',[10,10,400,300])
fplot(f,[0, 2*pi]); hold on;
scatter(x,t); hold on;
scatter(test_x,test_t);
%red - training data (x,t), yellow - test data (test_x,test_t)
```



3.3.2 same but with mu = 1

```
w = lsefit(x, t, B, @func_hat, 1);
f = @(s) point_hat(s, B)*w;

figure;
set(gcf,'position',[10,10,400,300])
```

```
fplot(f,[0, 2*pi]); hold on;
scatter(x,t); hold on;
scatter(test_x,test_t);
%red - training data (x,t), yellow - test data (test_x,test_t)
```



3.4 observations for hat basis

```
%with regularized least square, we are minimizing norm(w)^2/mu^2 in addition
%to the mean square error and the contribution of norm(w)^2/mu^2 is larger
%for smaller choice of mu. we see here that for smaller mu, e.g. mu = 1
%the norm(w)^2 contributes more to the cost, meaning it is putting a greater
%penalty for training larger parameters. In this sense, a smaller mu "draws
%the w vector towards the origin".

%Indeed, from the above figures, note that when mu is small, it means we
%scaled the hyperparameters up and so there is larger penalty for large w
%and so the parameters and thus also the prediction curve gets drawn towards
%the origin more and fluctuate less.

%another observation is that increasing the weight of hyperparameters
%(lowering mu) will make the regression line fit the training set more
%poorly (red points), but it can potentially make regression line predict
%the test_set (yellow points) better as it reduces overfitting.
```

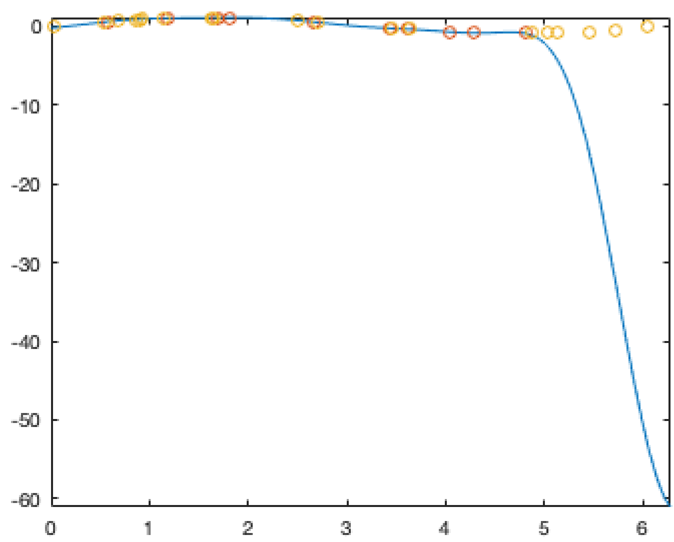
3.5.1 preparation, declare choice of basis (gaussian)

```
B = gauss_basis(0,2*pi,10);
```

3.5.2 gauss basis linear regression w/ #basis = 10, mu = 10^5

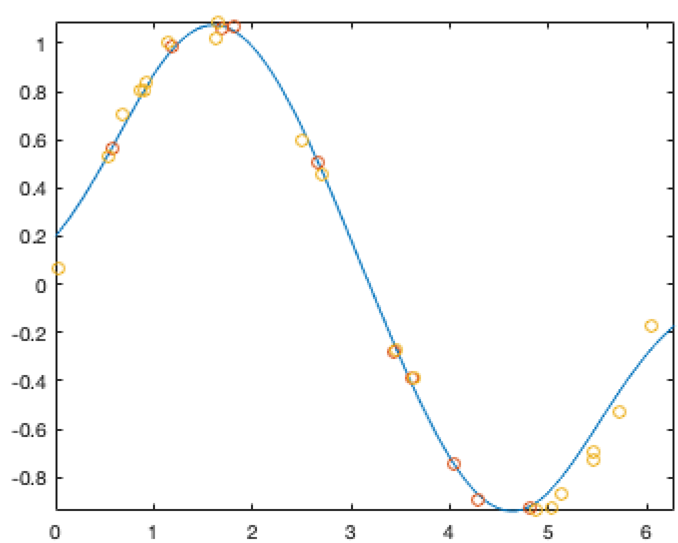
```
%learn predictor parameters
w = lsefit(x, t, B, @func_gauss, 10^5);

%use learned parameters w to predict t given x, by doing dot product
%with (phi_0(x),...,phi_m(x))
f = @(s) point_gauss(s, B)*w;
```

```
figure;
set(gcf,'position',[10,10,400,300])
fplot(f,[0, 2*pi]); hold on;
scatter(x,t); hold on;
scatter(test_x,test_t);
%red - training data (x,t), yellow - test data (test_x,test_t)
```



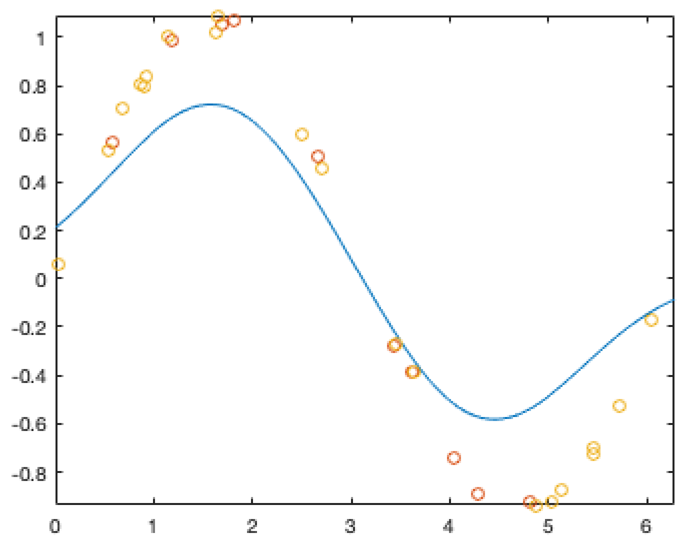
3.6.1 same as previous, but with hyperparameter mu = 10

```
w = lsefit(x, t, B, @func_gauss, 10);
f = @(s) point_gauss(s, B)*w;

figure;
set(gcf,'position',[10,10,400,300])
fplot(f,[0, 2*pi]); hold on;
scatter(x,t); hold on;
scatter(test_x,test_t);
%red - training data (x,t), yellow - test data (test_x,test_t)
```



3.6.2 same as previous, but with hyperparameter mu = 1

```
w = lsefit(x, t, B, @func_gauss, 1);
f = @(s) point_gauss(s, B)*w;

figure;
set(gcf,'position',[10,10,400,300])
fplot(f,[0, 2*pi]); hold on;
scatter(x,t); hold on;
scatter(test_x,test_t);
%red - training data (x,t), yellow - test data (test_x,test_t)
```

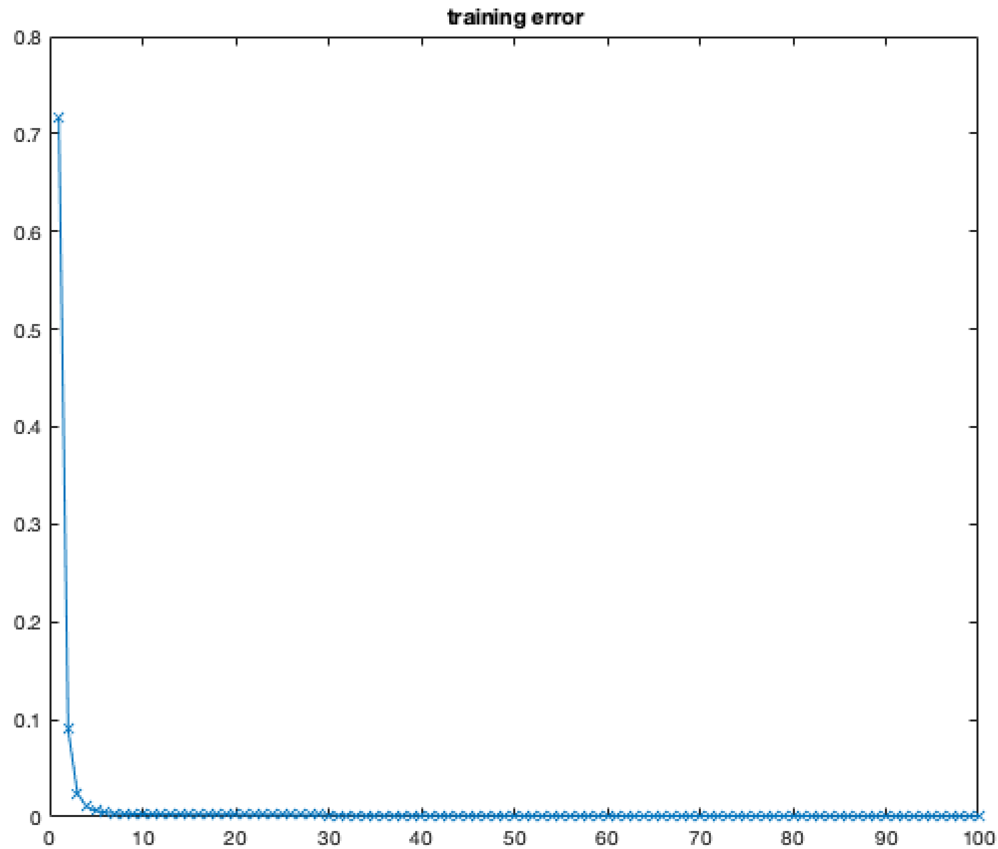


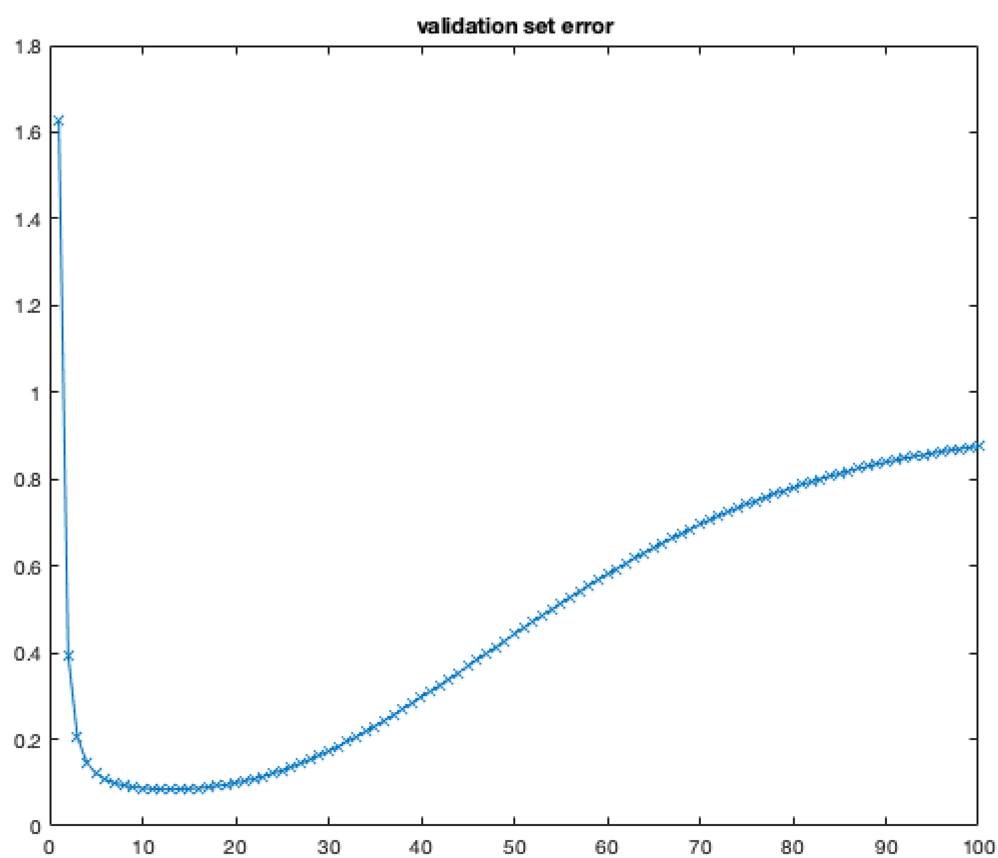
3.7 observations for gauss basis

```
%The hyperparameters in this case is also drawing the parameter vector w
%towards 0, and so we also see that increasing hyperparameter (lowering mu)
%will again cause the prediction line to be pulled towards 0.

%one nice difference & observation is with 10 gaussian basis function, it
%seems that with small hyperparameter (mu = 10^5), there's huge drop in the
%prediction curve towards the right of the red training set points (x=5) that
%it was trained on, which pulls it very far from the yellow test set points,
%and this suggests that some parameter in w may be trained to hold a large
%value in order to fit the red data points better. This is unlike the hat
%functions since for hat functions, even with mu = 10^5 it stays relatively
%controlled in the interval [0,2*pi]

%so as a result, here when we increase the hyperparameters to mu = 10 and then
%to mu = 1, we see the large penalty kick in with norm(w)^2 being very large
%in the cost if w itself is large. This shows in the plots for mu = 10 and
%mu = 1 as the right drop has been entirely corrected.
%
%
%
%
%
%padding
```

3.8 fix #basis = 10, vary hyperparameter mu from 1 to 100

```
B = gauss_basis(0,2*pi,10);

tr_err = zeros(100,1);
ts_err = zeros(100,1);

for i = 1:100
    w = lsefit(x, t, B, @func_gauss, i);
    Tr_X = point_gauss(x,B)*w;
    Ts_X = point_gauss(test_x,B)*w;
    tr_err(i) = norm(Tr_X-t)^2;
    ts_err(i) = norm(Ts_X-test_t)^2;
end

%plot squared error
figure;
set(gcf,'position',[10,10,600,480])
plot(1:100,tr_err,'-x');
title('training error');

figure;
set(gcf,'position',[10,10,600,480])
plot(1:100,ts_err,'-x');
title('validation set error');
```


training error
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
10
20
30
40
50
60
70
80
90
100


validation set error
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
0
10
20
30
40
50
60
70
80
90
100

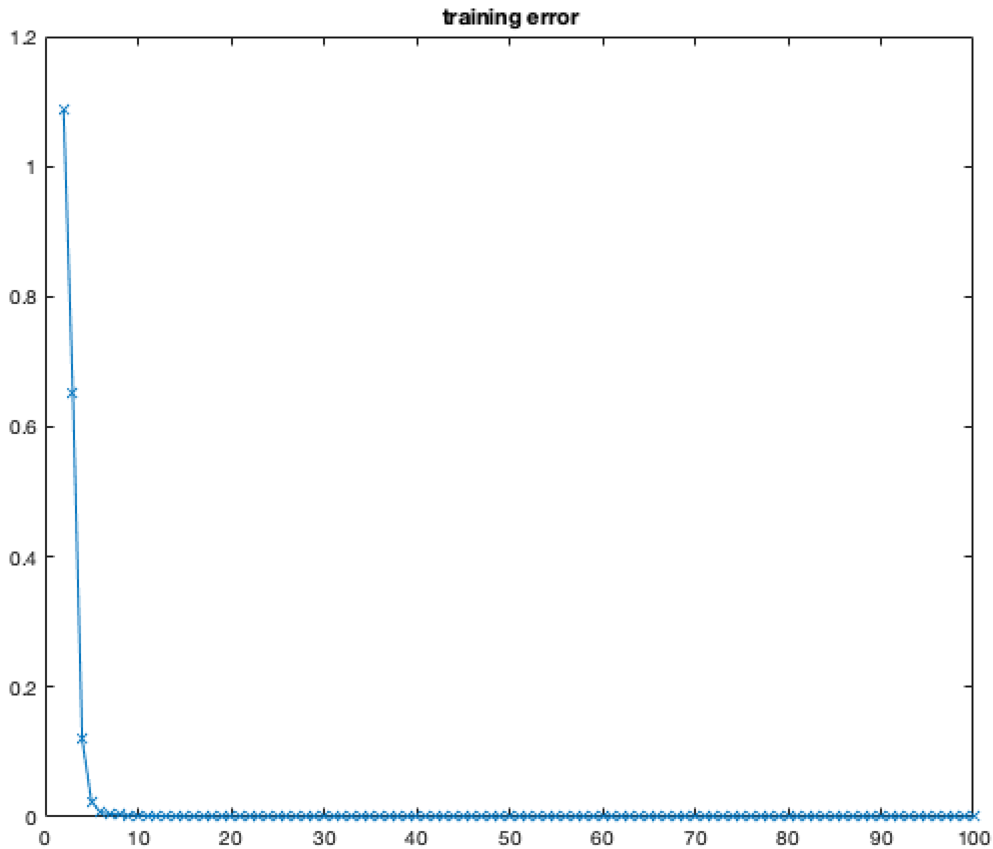
3.9 observation in validation error

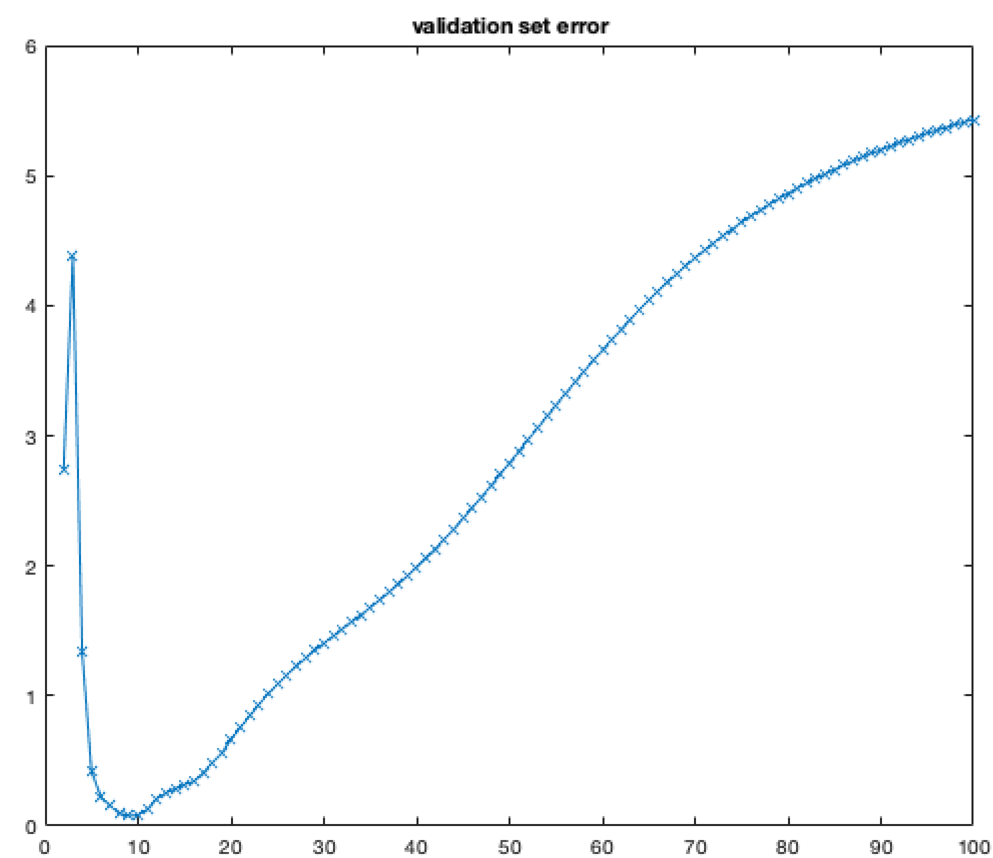
```
%to answer this, we look at the smallest element in the ts_err vector which
%contains in the ith element the total squared model error for observations
%in test_mat using mu = i.

%indeed, from the graph and below we confirm mu = 13 is the best hyperparemeter,
%and that makes sense because with too much regularization we see unnecessary
%dampening of prediction curve where too much information is lost to the
%regularization, and with too little the prediction becomes overly complex and
%overfits the training data, so while it performs just marginally better on
%the already well-fit training set, it fails on the test set.

[~, optimal_mu] = min(ts_err);
fprintf('optimal hyperparameter mu: %d\n', optimal_mu);
```

```
optimal hyperparameter mu: 13
```

3.10 fix hyperparameter mu = 13, vary #basis from 1 to 100

```
tr_err = zeros(100,1);
ts_err = zeros(100,1);

for i = 1:100
    B = gauss_basis(0,2*pi,i);
    w = lsefit(x, t, B, @func_gauss, 13);
    Tr_X = point_gauss(x,B)*w;
    Ts_X = point_gauss(test_x,B)*w;
    tr_err(i) = norm(Tr_X-t)^2;
    ts_err(i) = norm(Ts_X-test_t)^2;
end

%plot squared error
figure;
set(gcf,'position',[10,10,600,480])
plot(1:100,tr_err,'-x');
title('training error');


figure;
set(gcf,'position',[10,10,600,480])
plot(1:100,ts_err,'-x');
title('validation set error');

%Similarly, we see a middle ground occur here where 1 or 2 basis functions
%are clearly inadequate to explain the relationship between x and t but
%larger number of basis functions, e.g. 50 or 100 is far too complicated
%and tries to overfit the data and explain noise. As a result, they also
%perform poorly on test set because they incorrectly interpret the noise as
%meaningful information to interpolate.
```


training error
1.2
1
0.8
0.6
0.4
0.2
0
0
10
20
30
40
50
60
70
80
90
100


validation set error
6
5
4
3
2
1
0
0
10
20
30
40
50
60
70
80
90
100