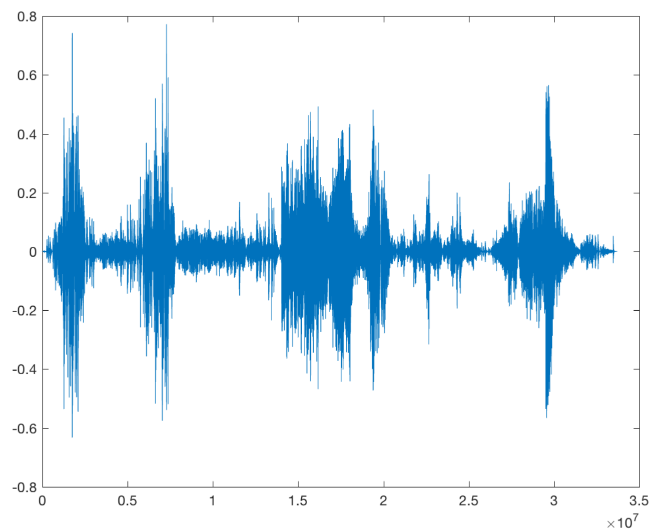
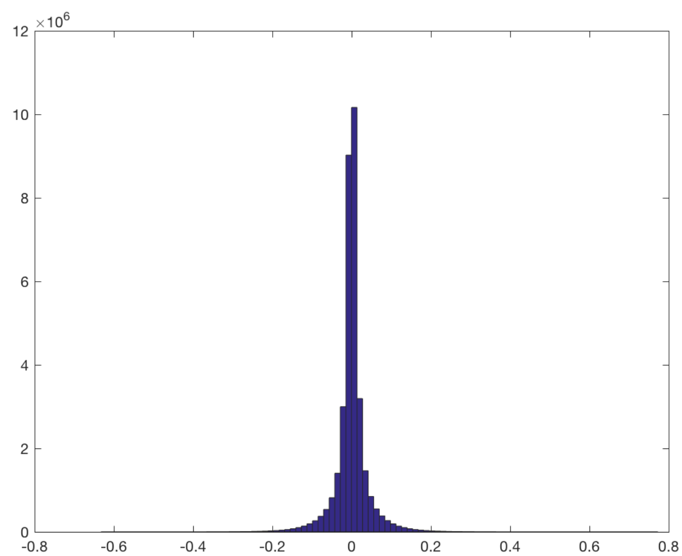
MLPR Assignment 1 – Predicting Audio
Chris Sipola (s1667278)
20 Oct 2016

Question 1

```
% Load data
load('/afs/inf.ed.ac.uk/group/teaching/mlprdata/audio/amp_data.mat')

% Plot amplitude data.
plot(amp_data);
saveas(gcf, 'Q1x_amplitudes.png');
```



```
hist(amp_data, 100);
saveas(gcf, 'Q1x_amplitude_hist.png');
```

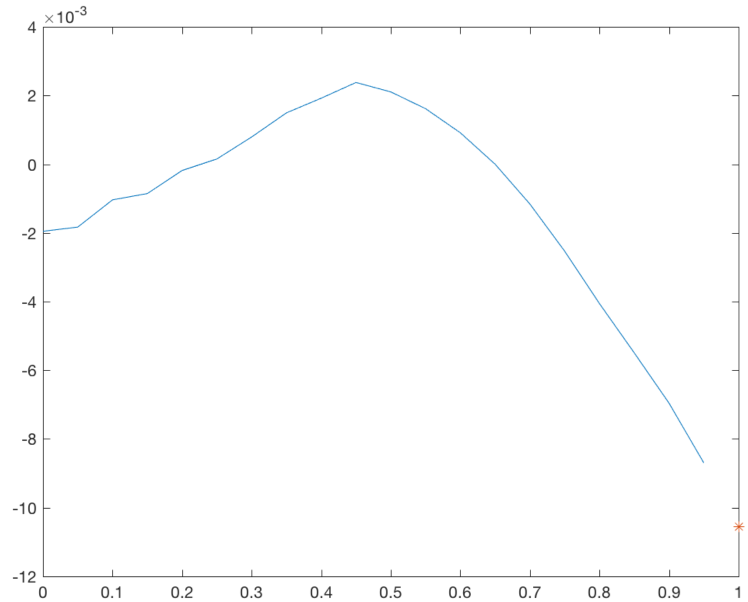


```
% Reshape data to wider form.
col_size = 21;
C = floor(size(amp_data, 1) / col_size);
amp_data = amp_data(1:(C * col_size));  % remove values that would
produce an incomplete row
amp_data = reshape(amp_data, col_size, C).';  % reshape is column-wise,
so need to switch dims and then transpose

% Get rows for train, val and test sets.
rng(287364823);  % for consistency of results
amp_data = amp_data(randperm(size(amp_data, 1)), :);  % shuffle rows
train_rows = 1:floor(0.7 * size(amp_data, 1));
val_rows = (max(train_rows) + 1):floor(0.85 * size(amp_data, 1));
test_rows = (max(val_rows) + 1):size(amp_data, 1);

% Get columns for X and y.
x_ids = 1:(col_size - 1);
y_ids = col_size;

% Separate into train, val and test sets.
X_shuf_train = amp_data(train_rows, x_ids);
y_shuf_train = amp_data(train_rows, y_ids);
X_shuf_val = amp_data(val_rows, x_ids);
y_shuf_val = amp_data(val_rows, y_ids);
X_shuf_test = amp_data(test_rows, x_ids);
y_shuf_test = amp_data(test_rows, y_ids);
```

Question 2

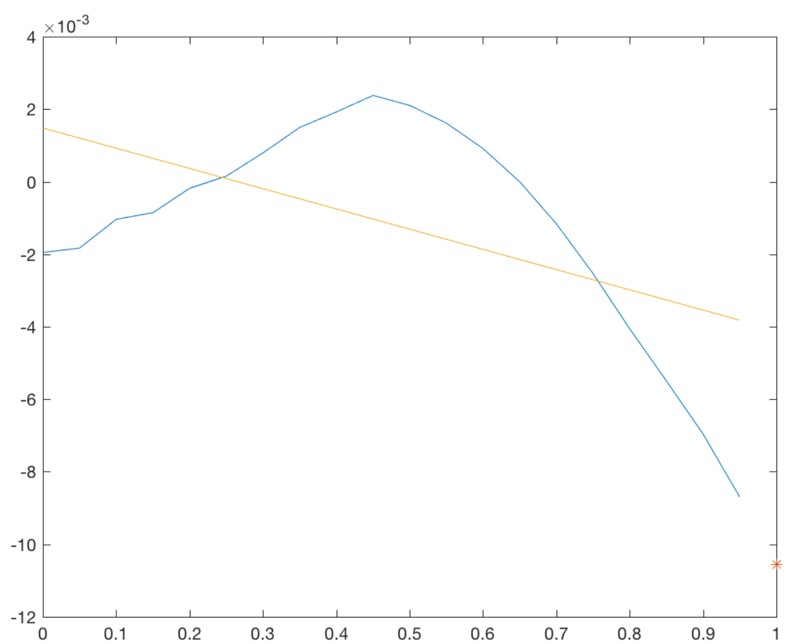
Part a

```
% Plot one row of x and y values in training set.
tt = (0:(1/20):(19/20)).';
row_to_plot = 2;  % just chose this because it looked nice; others
looked nice too
hold off;  % just in case I'm running code non-linearly...
plot(tt, X_shuf_train(row_to_plot, :));
hold on;
plot(1, y_shuf_train(row_to_plot), '*');  % plot t = 1 value as
asterisk
saveas(gcf, 'Q2a_one_row.png');
```



Part b

```
% Fit straight line to the 20 points.
Phi_1b = [ones(20, 1), tt];
w_fit = Phi_1b \ X_shuf_train(row_to_plot, :).';
ff = Phi_1b * w_fit;
plot(tt, ff);
saveas(gcf, 'Q2b_line_through_points.png');
```

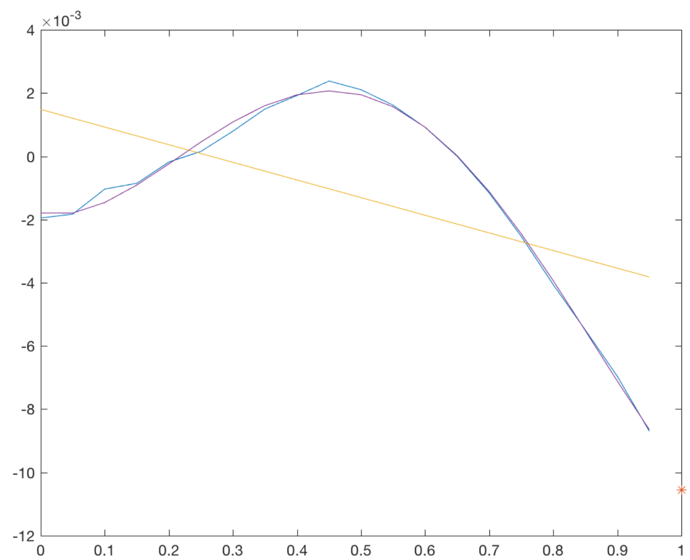


It may be better to use only the most recent two points for two reasons:

1. It would likely give a predicted value (at t = 1) close to the value at t = 19/20. This is good: because of autocorrelation, each amplitude is a good predictor of the next amplitude.
2. The line going through the last two amplitudes will likely point in the right general direction of the amplitude at t = 1 given how smooth the data usually is. E.g., if the amplitude at t = 1 is lower than the value at t = 19/20, then the line will usually guess this correctly.

Part c

```
% Fit quadratic polynomial.
Phi_1c = [ones(20, 1), tt, tt.^2, tt.^3, tt.^4];
w_fit = Phi_1c \ X_shuf_train(row_to_plot, :).';
ff = Phi_1c * w_fit;
plot(tt, ff);
saveas(gcf, 'Q2c_polynomial_through_points.png');
```



We may want to use a longer context with a polynomial fit in order to capture the shape of the curve. Using a short context length with a high-order polynomial can lead to overfitting and a wild prediction.

Many of the curves are quite smooth (although some seem to oscillate). My guess is that two types of models will (or could) work well: (1) one with both a short context length and a low-order polynomial (for reasons stated in Part b); and/or (2) one with a medium or long context length and a higher-order polynomial (at least to the third degree) for reasons just stated in the last paragraph.

Question 3

Part a

$$w = (\Phi^T\Phi)(\Phi^T x)$$

$$w^T = (\Phi^T x)^T(\Phi^T\Phi)^{-1}$$

$$w^T = x^T\Phi(\Phi^T\Phi)^{-1}$$

$$v^T x = w^T\phi(t=1)$$

$$x^T v = x^T\Phi(\Phi^T\Phi)^{-1}\phi(t=1)$$

$$v = \Phi(\Phi^T\Phi)^{-1}\phi(t=1)$$

Part b

```
% Construct C x K design matrix Phi.
Phi = make_Phi(5, 3, tt);  % first argument is C, second is K
vv = make_vv(Phi);

% Compare predictions to poly fit from before.
for X_train_row = 1:4

    w_fit = Phi_1c \ X_shuf_train(X_train_row, :).';
    phi1 = ones(size(Phi_1c, 2), 1);
    prediction1 = w_fit.' * phi1;

    vv = make_vv(make_Phi(20, 5, tt));  % C = 20, K = 5
    prediction2 = X_shuf_train(X_train_row, :) * vv;

    disp(strcat('row: ' + string(X_train_row)));
    disp(strcat('w^T * phi(t=1): ' + string(prediction1)));
    disp(strcat('v^T * x: ' + string(prediction2)));
    disp(' ')
end
```

Output:

```
row: 1
w^T * phi(t=1): -0.017183
v^T * x: -0.017183

row: 2
w^T * phi(t=1): -0.010021
```

```
v^T * x: -0.010021

row: 3
w^T * phi(t=1): -0.0072878
v^T * x: -0.0072878

row: 4
w^T * phi(t=1): -0.00082538
v^T * x: -0.00082538
```

Note that each gives equal value.

See "Appendix: Functions" section at the end of this document for `make_Phi` and `make_vv`.

Part c

```
% Evaluate various C and K. Not very efficient to calculate all test
results, but I find
% it cleaner.
C_max = 20;  % will test C from 1 to C_max
K_max = 5;  % will test K from K to min(C, K_max) (since I want K <= C)
E_train = Inf * ones(C_max, K_max);  % initialize with Inf b/c will
take min
E_val = Inf * ones(C_max, K_max);
E_test = Inf * ones(C_max, K_max);
for C = 1:C_max

    % Create data snippets of length C.
    C_idx = (20 - C + 1):20;
    X_shuf_train_C = X_shuf_train(:, C_idx);
    X_shuf_val_C = X_shuf_val(:, C_idx);
    X_shuf_test_C = X_shuf_test(:, C_idx);

    % For each K, create v and record error for training and
validation.
    for K = 1:min(C, K_max)
        vv = make_vv(make_Phi(C, K, tt));
        E_train(C, K) = mean((X_shuf_train_C * vv - y_shuf_train).^2);
        E_val(C, K) = mean((X_shuf_val_C * vv - y_shuf_val).^2);
        E_test(C, K) = mean((X_shuf_test_C * vv - y_shuf_test).^2);
    end
end

print_best_CK_and_error(E_train, 'train');
print_best_CK_and_error(E_val, 'val');

% Get test error value for best C, K.
test_error_3c = E_test(2, 2);
disp(strcat('Test error for optimal C, K: ' + string(test_error_3c)));
```

Output:

```
Best C, K in train set:
C = 2, K = 2 (error = 1.3466e-05)

Best C, K in val set:
C = 2, K = 2 (error = 1.3682e-05)

Test error for optimal C, K: 1.3665e-05
```

I tested C from 1 to 20 and K from 1 to min(5, C). (I capped K at C because I figured I shouldn't fit a *p* order polynomial through *n* > *p* + 1 points. For example, the highest order polynomial I want to draw through two points is a line.)

Below is the mean square error for the validation set for all C and K. (I also calculated the same for the test set.)

		K				
		1	2	3	4	5
C	1	0.03730				
	2	0.07690	0.01370			
	3	0.12480	0.02520	0.01930		
	4	0.17840	0.04000	0.02580	0.04530	
	5	0.23480	0.05910	0.03260	0.04600	0.12700
	6	0.29210	0.08190	0.04260	0.04620	0.10360
	7	0.34870	0.10840	0.05390	0.05150	0.08870
	8	0.40320	0.13800	0.06700	0.05940	0.08510
	9	0.45480	0.16970	0.08370	0.06640	0.08590
	10	0.50350	0.20230	0.10380	0.07400	0.08970
	11	0.54930	0.23490	0.12650	0.08490	0.09340
	12	0.59270	0.26690	0.15060	0.09930	0.09780
	13	0.63360	0.29840	0.17540	0.11600	0.10460
	14	0.67220	0.32910	0.20010	0.13510	0.11470
	15	0.70880	0.35880	0.22450	0.15630	0.12720
	16	0.74350	0.38770	0.24870	0.17790	0.14150
	17	0.77640	0.41580	0.27230	0.19990	0.15830
	18	0.80770	0.44310	0.29490	0.22220	0.17670
	19	0.83740	0.46960	0.31670	0.24430	0.19620
	20	0.86560	0.49530	0.33790	0.26540	0.21710

As shown in the output above, mean square error were as follows:

- Train:
  - Best C = 2 and K = 2

  - Mean square error: 1.3466e-05
- Validation:
  - Best C = 2 and K = 2
  - Mean square error: 1.3682e-05
- Test:
  - Using C = 2 and K = 2, test error is 1.3665e-05

Question 4

Part a

```
% Evaluate various C.
C_max = 20;
E_train = Inf * ones(C_max, 1);  % initialize error array
E_val = Inf * ones(C_max, 1);
for C = 1:C_max

    C_idx = (20 - C + 1):20;

    % Fit weights using training data.
    w_fit = X_shuf_train(:, C_idx) \ y_shuf_train;

    % Apply weights to both training and validation data.
    ff_train = X_shuf_train(:, C_idx) * w_fit;
    ff_val = X_shuf_val(:, C_idx) * w_fit;

    % Calculate error for training and validation error.
    E_train(C) = mean((ff_train - y_shuf_train).^2);
    E_val(C) = mean((ff_val - y_shuf_val).^2);

end

% get test error and best C in training set
[M, C] = min(E_train);

% Get test error for best C in validation set.
[M, C] = min(E_val);  % get best C
test_error_4b = E_test(C);
```

Context length 20 gives the lowest mean square error on the training set. This is simply because adding features decreases the training error due to overfitting. Context length 18 has the lowest mean square error in the validation set.

Part b

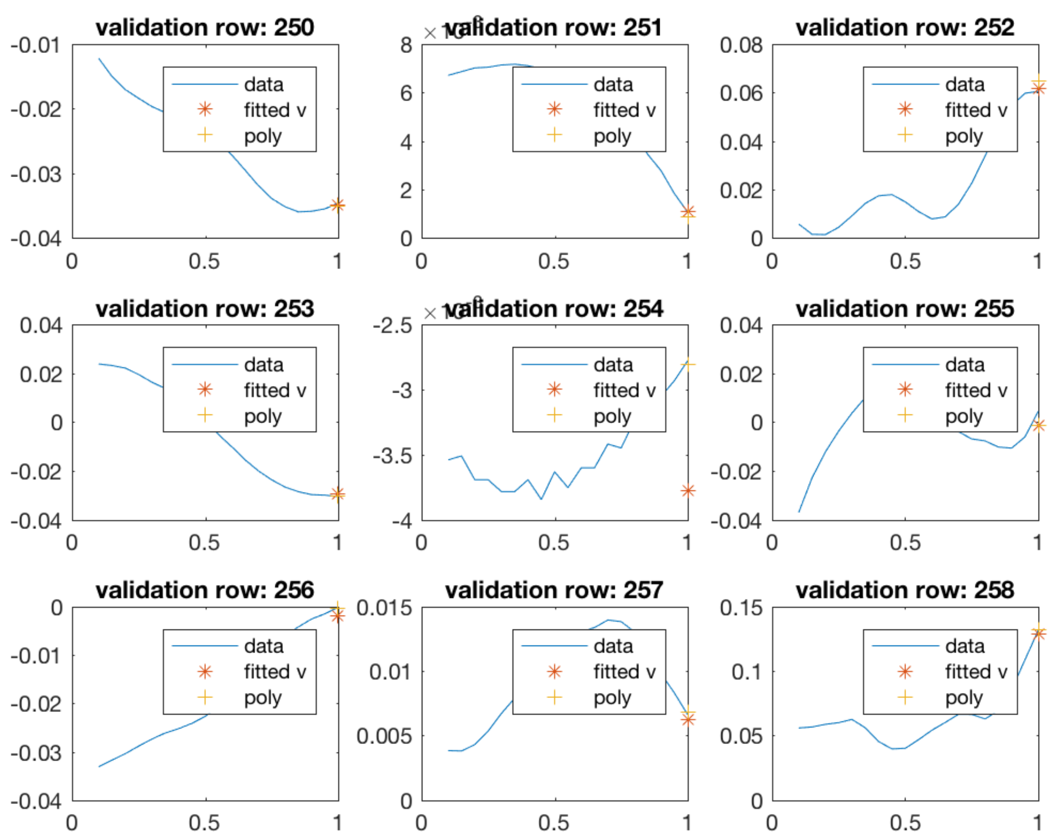
```
% Sloppy code to perform sanity check: look at some plots and
```

```
predictions.
num_plots = 9;
start_row = 250;
plot_num = 1;
grid_width = floor(sqrt(num_plots));

hold off;
for row_to_plot = start_row:(start_row + num_plots - 1)

    subplot(grid_width, grid_width, plot_num);

    % Pick row to plot and C
    C = 18;
    C_idx = (20 - C + 1):20;
    N_train = size(X_shuf_train, 1);
    % N_val = size(X_shuf_val, 1);

    % Find fit on training and get prediction for t = 1.
    w_fit = [ones(N_train, 1), X_shuf_train(:, C_idx)] \ y_shuf_train;
    ff_val = [1, X_shuf_val(row_to_plot, C_idx)] * w_fit;

    % Plot whole curve (including t = 1).
    hold off;
    plot([tt(C_idx); 1], [X_shuf_val(row_to_plot, C_idx),
y_shuf_val(row_to_plot)]);

    % Plot the prediction where we fitted v.
    hold on;
    plot(1, ff_val, '*');

    % Plot polynomial prediction.
    C = 2;
    K = 2;
    C_idx = (20 - C + 1):20;
    vv = make_vv(make_Phi(C, K, tt));
    ff_val2 = X_shuf_val(row_to_plot, C_idx) * vv;
    plot(1, ff_val2, '+');
    legend('data', 'fitted v', 'poly');
    title(strcat('validation row: ' + string(row_to_plot)))

    plot_num = plot_num + 1;
end
saveas(gcf, 'Q4b_grid_plot_sanity_check.png');

% Plot training error and validation error against C.
hold off;
plot(1:size(E_train, 1), E_train);
hold on;
plot(1:size(E_train, 1), E_val);
title('Training & validation error vs. C');
xlabel('C');
ylabel('error');
legend('E_{train}','E_{val}');
saveas(gcf, 'Q4b_training_vs_error.png');

% Compare to best C to best polynomial from question 3c.
```

```
test_error_4b / test_error_3c
```

The mean square error is about 59 times that of the of the polynomial model (8.0572e-04 vs. the polynomial mean square error of 1.3665e-05).

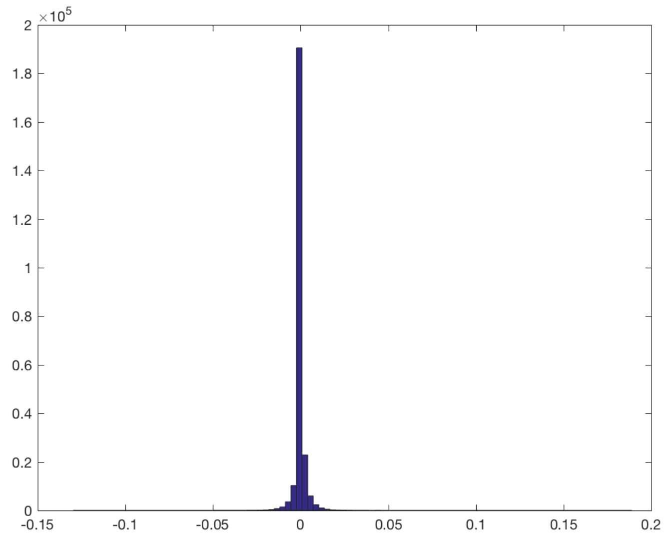
As a sanity check, I plotted predictions for both models for some example rows in the validation set. The point labeled “fitted v” refers to the prediction made by the model where we fitted *v* directly; “poly” is for the polynomial model. The “fitted v” model can be very wrong when the data is noisy. (This grid of plots only shows one where this is the case, but I saw it elsewhere as well.)

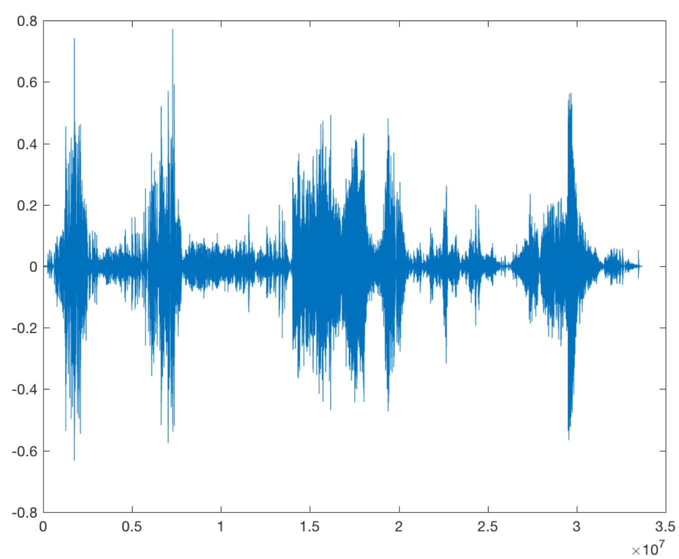


Part d

```
% Plot hist of residuals on validation data

% Calculate residuals.
C = 2;
K = 2;
C_idx = (20 - C + 1):20;
```

```
X_shuf_val_C = X_shuf_val(:, C_idx);
vv = make_vv(make_Phi(C, K, tt));
residuals = X_shuf_val_C * vv - y_shuf_val;

% Plot residuals.
hist(residuals, 100);
saveas(gcf, 'Q4c_residuals_hist.png');
2 * std(residuals);
```





Nearly all residuals seem to fall within the 0 ± 0.005 range, which is small when compared to the range of amplitudes. The values at the tails of the histogram (i.e., the largest errors) are likely predictions for points that occur in the noisiest areas of the file.

Question 5

Another way to make predictions:

- Make the design matrix Phi consist of K lags of x. For example, for K = 5, the design matrix could look like this (where, say, "x5" is *x* at t = 5/20):

x4	x3	x2	x1	x0
x5	x4	x3	x2	x1
x6	x5	x4	x3	x2
x7	x6	x5	x4	x3
x8	x7	x6	x5	x4
x9	x8	x7	x6	x5
x10	x9	x8	x7	x6
x11	x10	x9	x8	x7
x12	x11	x10	x9	x8
x13	x12	x11	x10	x9
x14	x13	x12	x11	x10
x15	x14	x13	x12	x11
x16	x15	x14	x13	x12
x17	x16	x15	x14	x13
x18	x17	x16	x15	x14
x19	x18	x17	x16	x15
x20	x19	x18	x17	x16

- The target vector would be values of *x* from t = 5/20 to t = 21/20.
- We could try a bias term as well.
- We also wouldn't be restrained to snippets (e.g., C = 20) using this method.

Other things to try:

- Fit RBFs, maybe starting with five functions for the 20 points.
- I wouldn't have very high hopes for regularization with the polynomial models since it's time series data (we want the prediction to be in the vicinity of value right before it). But I would try regularization on the model where we are fitting *v* directly.

Appendix: Functions

```
function Phi = make_Phi(C, K, tt)
    Phi = zeros(C, K);
    tt_short = tt((20 - C + 1):20);
    for k = 1:K
        Phi(:, k) = tt_short.^(k-1);
    end
end

function vv = make_vv(Phi)
    phi1 = ones(size(Phi, 2), 1);
    vv = Phi * inv(Phi.' * Phi).' * phi1;
end

function nada = print_best_CK_and_error(E, set_type)
    [M, I] = min(E(:));
    [C, K] = ind2sub(size(E), I);
```

```
    disp(strcat('Best C, K in ' + string(set_type) + ' set:'));
    disp(strcat('C = ' + string(C) + ', K = ' + string(K) + ' (error = ' +
string(M) + ')'));
    disp(' ');
end
```