jgiebas_HW3

February 5, 2018

1 46-926, Statistical Machine Learning 1: Homework 3

Author : Jordan Giebas *Due Date*: Februrary 5th, 2018
Sorry in advance, the %%capture magic function wasn't working for me so there are a plethora of error messages

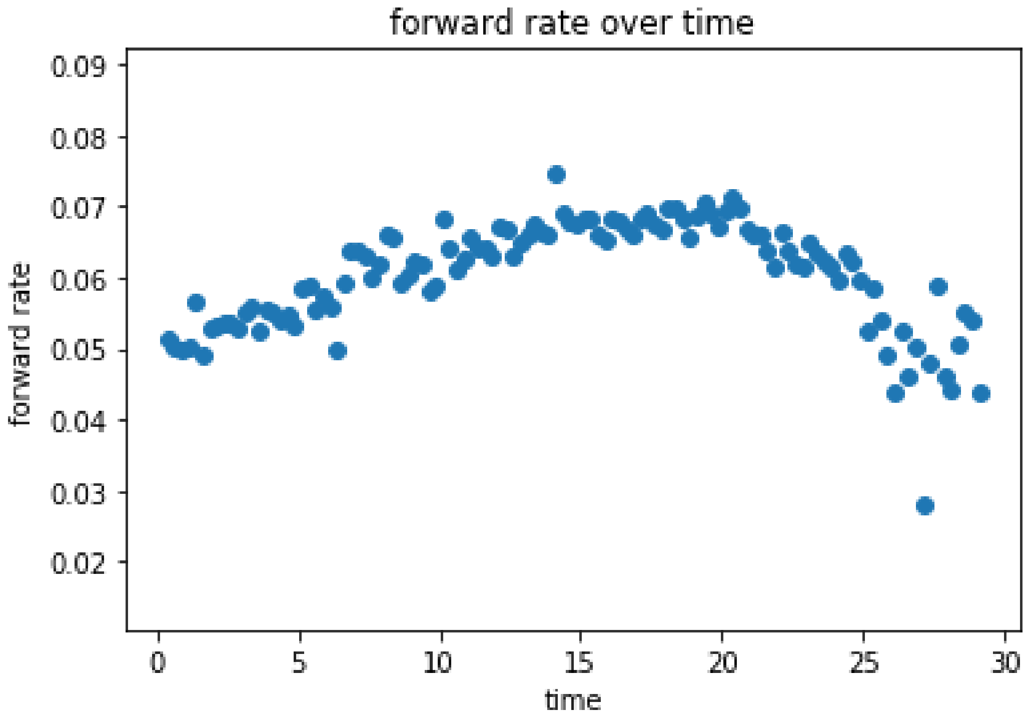
1.1 Question 1

1.1.1 Part (a)

Load the forward rate data, plot the empirical forward rate data against time.

```
In [1]: import warnings
        warnings.filterwarnings(action='ignore')

        import numpy as np
        import pandas as pd

        import matplotlib
        import matplotlib.pyplot as plt
        %matplotlib inline
        #Set the default figure size for the notebook, so they are not all tiny
        matplotlib.rcParams['figure.figsize'] = (12,8)

In [2]: df = pd.read_csv('forward_rates.csv')
        df.head()

Out[2]:      time      rate
        0  0.3699  0.051389
        1  0.6219  0.050082
        2  0.8740  0.049707
        3  1.1260  0.050316
        4  1.3699  0.056612

In [3]: plt.xlabel("time")
        plt.ylabel("forward rate")
        plt.title("forward rate over time")
        plt.scatter(df.time, df.rate)
```

Out[3]: <matplotlib.collections.PathCollection at 0x1090a6400>



There seems to be a lot of noise around a generally obvious curve. I agree, that linear/quadratic functions wouldn't model this well.

1.1.2 Part (b)

Fit a smoothing spline to the data using `LinearGAM` as seen in class, and cross-validate for a reasonable λ. Superimpose a plot of the fit onto the plot above. How well is the fit?

In [4]:
```
!pip3 install pygam
```

```
Requirement already satisfied: pygam in /usr/local/lib/python3.6/site-packages
Requirement already satisfied: progressbar2 in /usr/local/lib/python3.6/site-packages (from pyg
Requirement already satisfied: future in /usr/local/lib/python3.6/site-packages (from pygam)
Requirement already satisfied: numpy in /usr/local/lib/python3.6/site-packages (from pygam)
Requirement already satisfied: scipy in /usr/local/lib/python3.6/site-packages (from pygam)
Requirement already satisfied: python-utils>=2.1.0 in /usr/local/lib/python3.6/site-packages (:
Requirement already satisfied: six in /usr/local/lib/python3.6/site-packages (from python-utils
```

In [5]:
```
%%capture
from pygam import LinearGAM
gam = LinearGAM().gridsearch(df.time,df.rate,lam = np.logspace(-5, 3, 30))
```

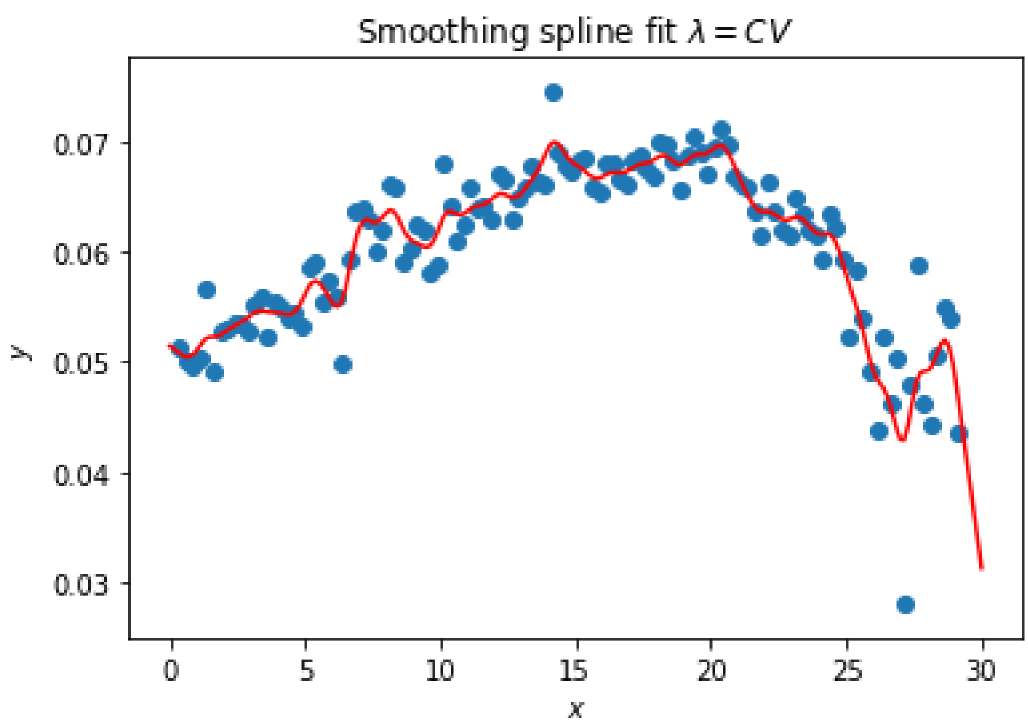
```
In [6]: cv_lam = gam.lam
        print("Cross-validated (default) lambda: ", cv_lam)
```

```
Cross-validated (default) lambda:  0.9236708571873865
```

```
In [7]: %%capture
        gam = LinearGAM(n_splines=len(df.index)+1, lam=cv_lam).fit(df.time,df.rate)
```

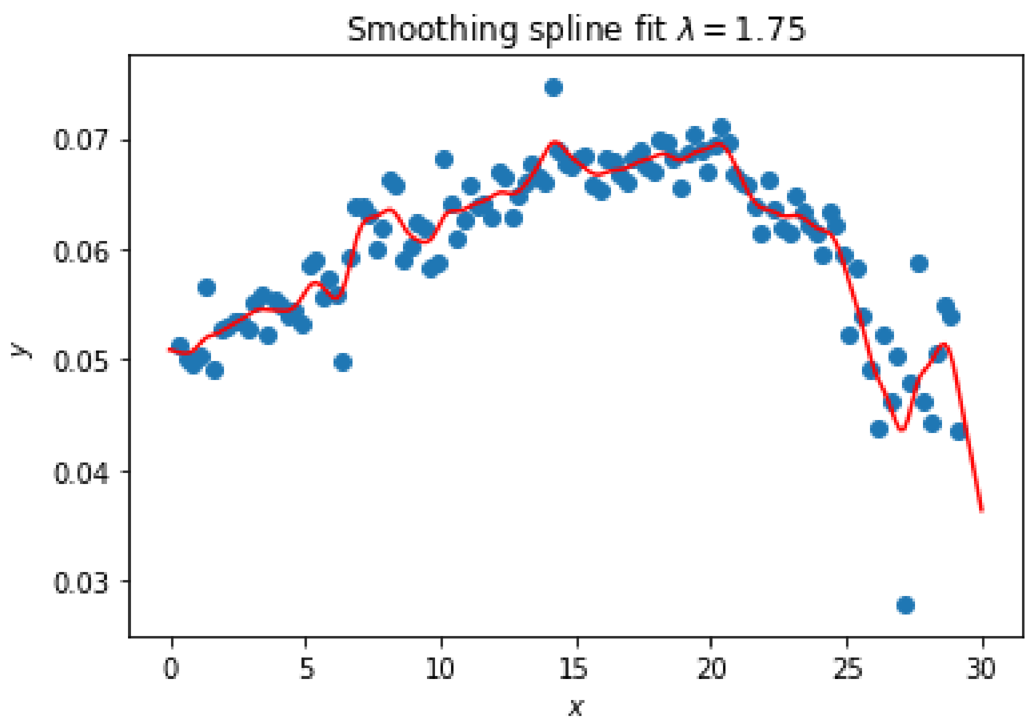
```
In [8]: # Define the domain over which spline will predict
        # Many points for smoothness
        x_grid = np.linspace(0,30,num=200)

        fig, ax = plt.subplots()
        ax.plot(x_grid, gam.predict(x_grid), 'r-')
        ax.scatter(df.time,df.rate)
        ax.set(xlabel = r"$x$",
               ylabel = r"$y$",
               title = r"Smoothing spline fit $\lambda = CV$");
        plt.show()
```

```
/usr/local/lib/python3.6/site-packages/pygam/utils.py:165: UserWarning: Expected 2D input data
  warnings.warn(msg)
```



The smoothing spline does okay, and captures the variability where present but it is still very wiggly. Below I try a different value of lambda to penalize curvature more.

```
In [9]: %%capture
        gam_2 = LinearGAM(n_splines=len(df.index)+1, lam=1.75).fit(df.time,df.rate)

In [10]: fig, ax = plt.subplots()
         ax.plot(x_grid, gam_2.predict(x_grid), 'r-')
         ax.scatter(df.time,df.rate)
         ax.set(xlabel = r"$x$",
                ylabel = r"$y$",
                title = r"Smoothing spline fit $\lambda = 1.75$");
         plt.show()

/usr/local/lib/python3.6/site-packages/pygam/utils.py:165: UserWarning: Expected 2D input data
  warnings.warn(msg)
```



Honestly, I think this is better with $\lambda = 1.75$. "Better" in the sense that it's not as wiggly but it still many of the features.

1.1.3 Part (c)

Cubic splines are nothing but piecewise cubic functions between various "knots" (datum), with some additional constraints that at each knot they are continuous, and match regarding their first and second derivatives. It's easy to see that integrating cubics over a defined interval is simply, and a closed-form solution exists. Since the first and second derivatives align at each knot, or element in the partition, one can compute the integral between adjacent elements in the partition and then sum them up to acquire the entire integral.

1.2 Question 2: Kernel Regression and varying smoothness

Below is the chunk of code from `adaptivity_starter.py`

In [11]:
```
import numpy as np
import pandas as pd
import scipy.stats

### Starter code for Homework 2, problem 7 ###

#Our true mean function: will be sin(x/2) on [0,4*pi] and sin(6*x) on [4*pi,8*pi]
def mu(x):

    #Initialize a vector of zeros
    y = np.zeros(len(x))

    #Figure out which points are to the left or right of 4*pi
    left_points = (x<=4*np.pi);
    right_points = (x>4*np.pi)

    #Assign the appropriate sine values
    y[left_points] = np.sin(x[left_points]/2.0);
    y[right_points] = np.sin(6*x[right_points])

    #Return y
    return(y)


#A function to draw a sample from this curve
def generate_sample(n):

    # Define x
    x = np.random.uniform(low=0, high=8*np.pi, size=n)
    x.sort()

    #Sample the y coordinates as gaussians around mu(x)
    y = mu(x) + np.random.normal(loc=0, scale=0.2, size=n)

    #Bind this all together into a pandas data frame
    return(pd.DataFrame({'x':x, 'y':y}))


#We set the seed so that your homeworks will match
scipy.random.seed(7)
#Sample 300 points.  This is your data set!
data = generate_sample(500)
```

In [12]:
```
data.tail()
```

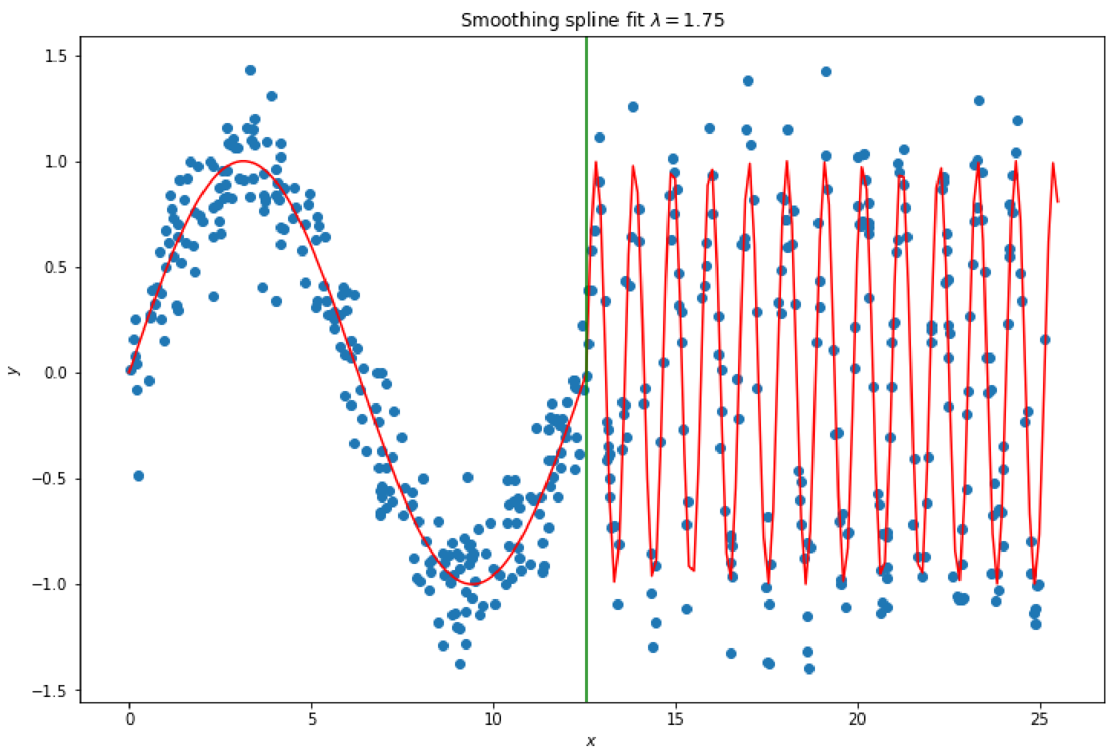
Out[12]:

```
             x         y
495  24.871418 -1.114434
496  24.879256 -1.184319
497  24.926871 -1.008500
498  24.945696 -0.999930
499  25.112844  0.158893
```

In [13]:

```python
%matplotlib inline

matplotlib.rcParams['figure.figsize'] = (12,8)
fig, ax = plt.subplots()

# Plot true mu(x)
x_grid = np.linspace(start=0, stop = 25.5, num=200)
y_grid = mu(x_grid)
ax.plot(x_grid, mu(x_grid), 'r-')

# Superimpose scatterd data
ax.scatter(data.x,data.y)
ax.set(xlabel = r"$x$",
       ylabel = r"$y$",
       title = r"Smoothing spline fit $\lambda = 1.75$");

# Put a vertical line to demonstrate where 4\pi is
plt.axvline(x=4*np.pi, color='g')
```

Out[13]: <matplotlib.lines.Line2D at 0x10c7d2c18>

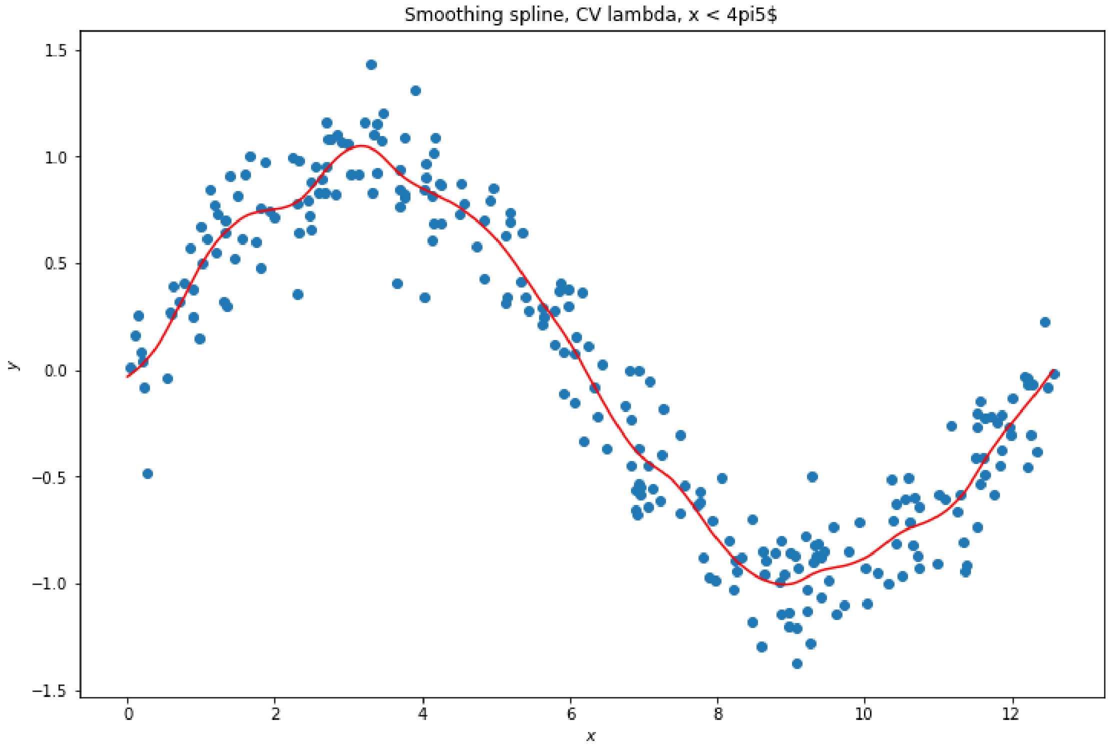


The blue dots in the plot above represent the data, and we can see that the true $\mu(x)$ fits this data quite well. The interesting facet of the data is at $x = 4\pi$, where the green vertical line occurs. $\forall x \leq 4\pi$, it would be sensible to describe the data using a sinusoid. $\forall x > 4\pi$, it seems like we may be able to describe the data with a sinusoid once again but the frequency with which the crests and troughs oscillate is quite intensified.

1.2.1 Part (b)

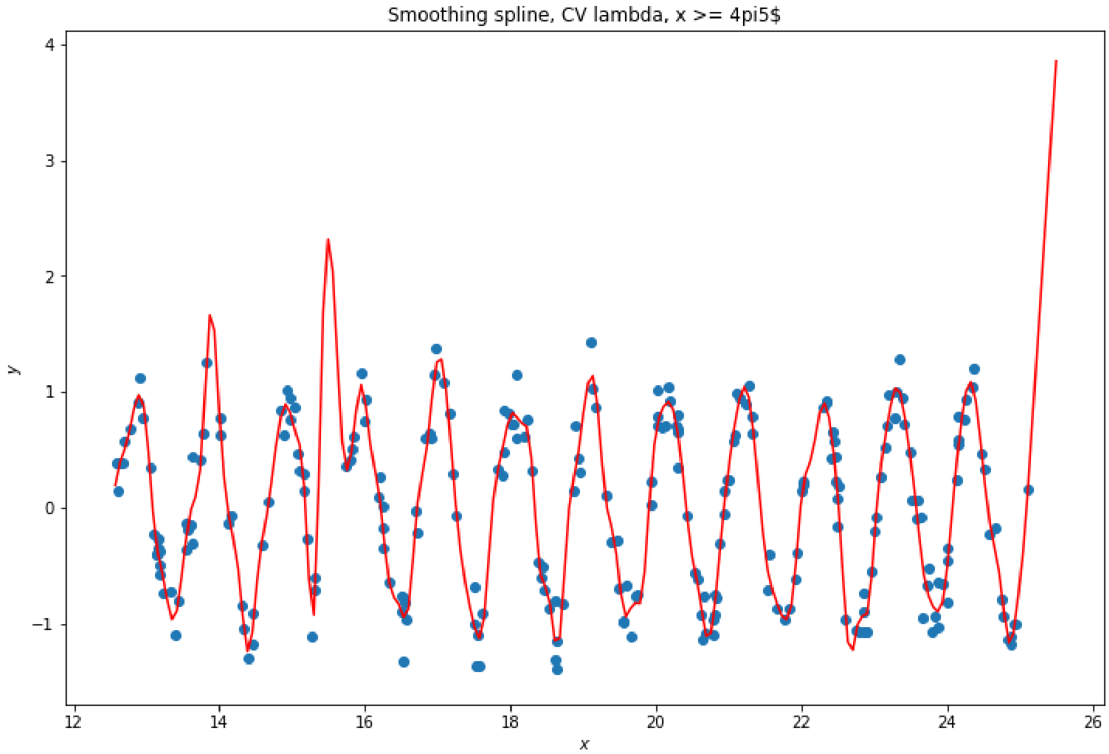
Fit a smoothing spline for,
$\forall x \leq 4\pi$
$\forall x > 4\pi$
All the points

```
In [14]: le4pi_data = data[data.x <= 4*np.pi]
         gt4pi_data = data[data.x >  4*np.pi]

In [15]: len(le4pi_data.index) + len(gt4pi_data.index) == len(data.index)

Out[15]: True

In [16]: %%capture
         ## Smoothing Spline for x \in [0,4\pi] with suggested lam grid ##

         # Fit model
         gam_2b_le4pi = LinearGAM().gridsearch(le4pi_data.x, le4pi_data.y, lam = np.logspace(-5
         cv_lam_2b_le4pi = gam_2b_le4pi.lam
         gam_2b_le4pi = LinearGAM(n_splines=100, lam=cv_lam_2b_le4pi).fit(le4pi_data.x,le4pi_da

In [17]: # Plotting
         matplotlib.rcParams['figure.figsize'] = (12,8)
         x_grid = np.linspace(0,4*np.pi,num=200)
         fig, ax = plt.subplots()
         ax.plot(x_grid, gam_2b_le4pi.predict(x_grid), 'r-')
         ax.scatter(le4pi_data.x,le4pi_data.y)
         ax.set(xlabel = r"$x$",
                ylabel = r"$y$",
                title = r"Smoothing spline, CV lambda, x < 4pi5$");

/usr/local/lib/python3.6/site-packages/pygam/utils.py:165: UserWarning: Expected 2D input data
  warnings.warn(msg)
```




```
In [18]: %%capture
         ## Smoothing Spline for x \in [4\pi:] with suggested lam grid ##

         # Fit model
         gam_2b_gt4pi = LinearGAM().gridsearch(gt4pi_data.x, gt4pi_data.y, lam = np.logspace(-5
         cv_lam_2b_gt4pi = gam_2b_gt4pi.lam
         gam_2b_gt4pi = LinearGAM(n_splines=100, lam=cv_lam_2b_gt4pi).fit(gt4pi_data.x,gt4pi_da

In [19]: x_grid = np.linspace(4*np.pi, 25.5,num=200)
         fig, ax = plt.subplots()
         ax.plot(x_grid, gam_2b_gt4pi.predict(x_grid), 'r-')
         ax.scatter(gt4pi_data.x,gt4pi_data.y)
         ax.set(xlabel = r"$x$",
                ylabel = r"$y$",
                title = r"Smoothing spline, CV lambda, x >= 4pi5$");
```

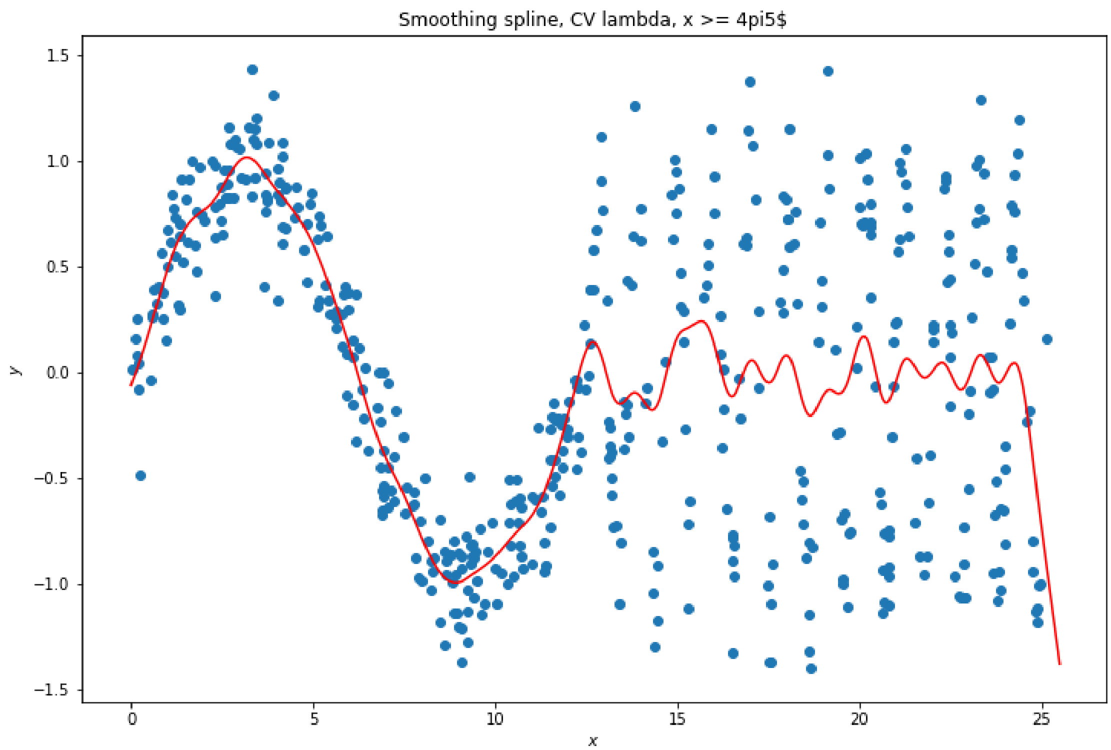
```
/usr/local/lib/python3.6/site-packages/pygam/utils.py:165: UserWarning: Expected 2D input data
  warnings.warn(msg)
```




```
In [20]: %%capture
         ## Smoothing Spline for x with suggested lam grid ##

         # Fit model
         gam_2b_alldata = LinearGAM().gridsearch(data.x, data.y, lam = np.logspace(-5, 4, 100))
         cv_lam_2b_alldata = gam_2b_alldata.lam
         gam_2b_alldata = LinearGAM(n_splines=100, lam=cv_lam_2b_alldata).fit(data.x,data.y)
```

```
In [21]: x_grid = np.linspace(0, 25.5,num=400)
         fig, ax = plt.subplots()
         ax.plot(x_grid, gam_2b_alldata.predict(x_grid), 'r-')
         ax.scatter(data.x,data.y)
         ax.set(xlabel = r"$x$",
                ylabel = r"$y$",
                title = r"Smoothing spline, CV lambda, x >= 4pi5$");
```

```
/usr/local/lib/python3.6/site-packages/pygam/utils.py:165: UserWarning: Expected 2D input data
  warnings.warn(msg)
```



The cell below prints out each of the Cross-Validated (optimal) lambdas

```
In [22]: print("CV_lambda_le4pi: ", cv_lam_2b_le4pi)
         print("CV_lambda_gt4pi: ", cv_lam_2b_gt4pi)
         print("CV_lambda_data : ", cv_lam_2b_alldata)
```

```
CV_lambda_le4pi:  23.10129700083163
CV_lambda_gt4pi:  0.0003511191734215131
CV_lambda_data :  8.111308307896872
```

The output of the above cell is sensible. For $x \in [0, 4\pi]$, $\lambda_{le4\pi}$ is very large. This is because we are penalizing for extreme curvature, which is sensible since this piece of the graph doesn't suggest much curvature. On the contrary, for $x \in [4\pi, 25.5]$, $\lambda_{gt4\pi}$ is very small. This again is sensible since the graph has a lot of curvature and we should therefore not penalize for a lot of curvature.

Lastly, the λ_{CV} for the whole domain of x is in between these values. In order to compensate for the lack of curvature in the first partition to the second, or vice-versas to adjust for the difference in curvature from the second partition to the first, the spline chooses a lambda in between the extreme cases we saw in each of the partitions separately.

This is a limitation of the smoothing splines: abrupt changes of curvature at any given knot will result in a case as the above due to the constraints in continuity, differentiability, and curvature at each knot.

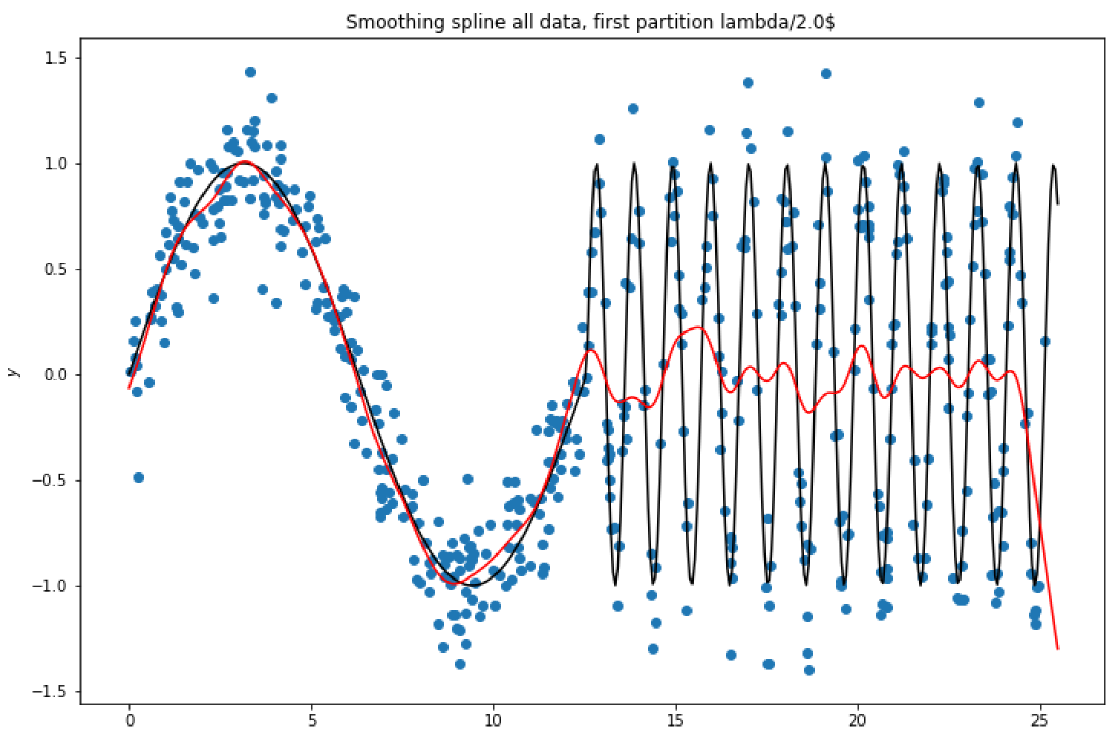
1.2.2 Part (c)

Using the above lambdas, refit the smoothing spline on the entire data set and superimpose the plot with the true mean $\mu(x)$ and the datum.

```
In [23]: %%capture
         # Case I: Lambda equals CV_lambda_le4pi/2 (23.10129700083163/2)
         gam_2c_1 = LinearGAM(n_splines=100, lam=cv_lam_2b_le4pi/2.0).fit(data.x,data.y)
```

```
In [24]: x_grid = np.linspace(0, 25.5,num=400)
         fig, ax = plt.subplots()
         ax.plot(x_grid, mu(x_grid), 'k-')
         ax.plot(x_grid, gam_2c_1.predict(x_grid), 'r-')
         ax.scatter(data.x,data.y)
         ax.set(xlabel = r"$x$",
                ylabel = r"$y$",
                title = r"Smoothing spline all data, first partition lambda/2.0$");
```

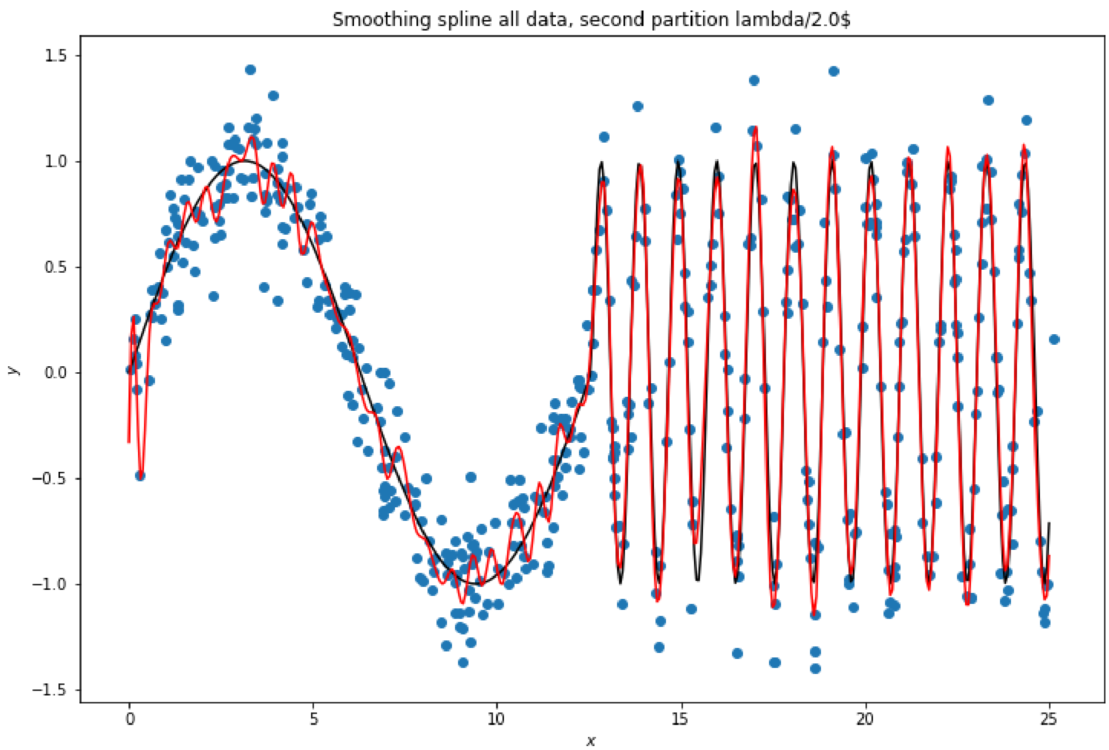
```
/usr/local/lib/python3.6/site-packages/pygam/utils.py:165: UserWarning: Expected 2D input data
  warnings.warn(msg)
```




```
In [25]: %%capture
         # Case II: Lambda equals CV_lambda_gt4pi/2 (0.00035111917342151 31/2)
         gam_2c_2 = LinearGAM(n_splines=100, lam=cv_lam_2b_gt4pi/2.0).fit(data.x,data.y)
```

```
In [26]: x_grid = np.linspace(0, 25.0,num=400)
         fig, ax = plt.subplots()
         ax.plot(x_grid, mu(x_grid), 'k-')
         ax.plot(x_grid, gam_2c_2.predict(x_grid), 'r-')
         ax.scatter(data.x,data.y)
         ax.set(xlabel = r"$x$",
                ylabel = r"$y$",
                title = r"Smoothing spline all data, second partition lambda/2.0$");
```

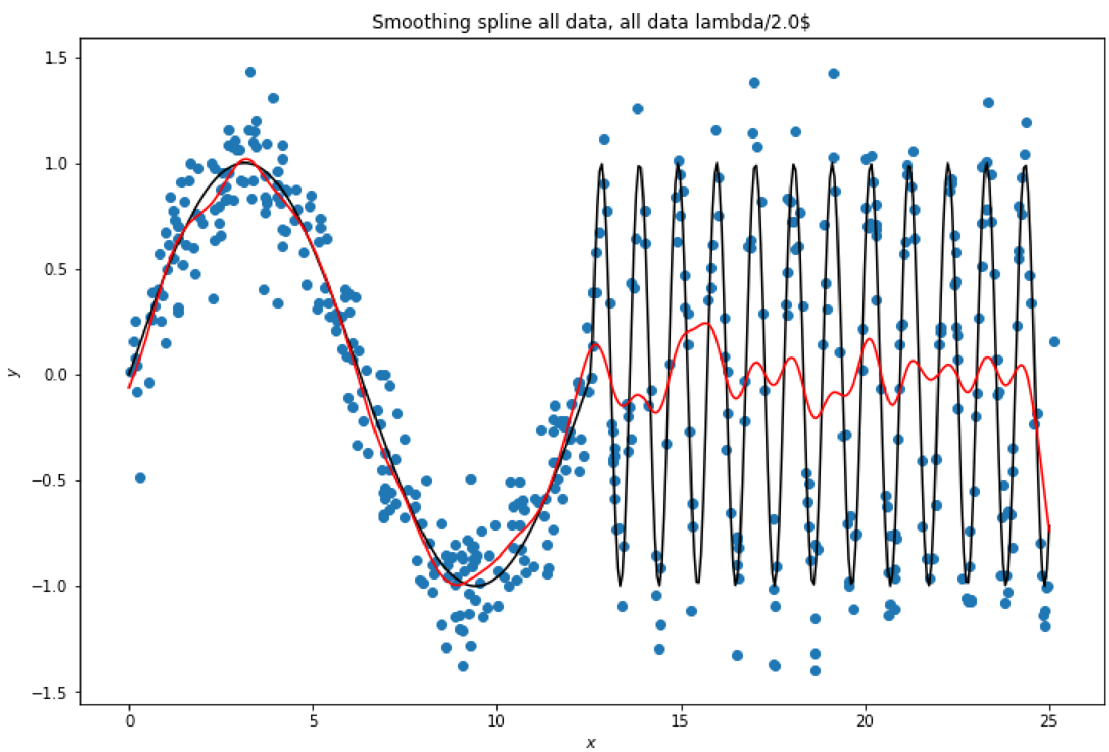
```
/usr/local/lib/python3.6/site-packages/pygam/utils.py:165: UserWarning: Expected 2D input data
  warnings.warn(msg)
```




```
In [27]: %%capture
         # Case III: Lambda equals CV_lambda_gt4pi/2 (0.0003511191734215131/2)
         gam_2c_3 = LinearGAM(n_splines=100, lam=cv_lam_2b_alldata).fit(data.x,data.y)
```

```
In [28]: x_grid = np.linspace(0, 25.0,num=400)
         fig, ax = plt.subplots()
         ax.plot(x_grid, mu(x_grid), 'k-')
         ax.plot(x_grid, gam_2c_3.predict(x_grid), 'r-')
         ax.scatter(data.x,data.y)
         ax.set(xlabel = r"$x$",
                ylabel = r"$y$",
                title = r"Smoothing spline all data, all data lambda/2.0$");
```

```
/usr/local/lib/python3.6/site-packages/pygam/utils.py:165: UserWarning: Expected 2D input data
  warnings.warn(msg)
```



In each of the three cases, I have already somewhat discussed the issue above.

When the λ_{CV} is very big, we are penalizing a lot for curvature. So we see that the first partition of the graph is well fit, but we don't capture the chaotic nature of the second partiiton at all. We can see that we are far from the true mean.

When the λ_{CV} is very small, we are not penalizing a lot for curvature. So we see that while we genearlly fit the first partition of the graph, we are not nearly smooth enough as the data suggests. This being said, we capture the essence of the second partition very well since we are more flexible for curvature. Over the first partition, we generally deviate quite a big from the mean, but over the second we right in line.

When the λ_{CV} is in between these two cases, we see that we are compromising to some degree. We generally are aligned with the mean over the first partition (minus some small deviations), but we do not model the mean/datum well at all over the second partiiton because we are underestimating.

1.3 Question 3

```
In [29]: train_df = pd.read_csv('bike_train.csv')
         test_df = pd.read_csv('bike_test.csv')
         test_df.head()
```

```
Out[29]:    dayofyear  dayofweek  workday  weathertype      temp  humidity  windspeed  \
         0          0          0        0            1  0.370000  0.692500   0.192167
         1          1          1        0            1  0.273043  0.381304   0.329665
         2          2          2        1            1  0.150000  0.441250   0.365671
         3          3          3        1            2  0.107500  0.414583   0.184700
         4          4          4        1            1  0.265833  0.524167   0.129987

            rentals
         0     2294
         1     1951
         2     2236
         3     2368
         4     3272
```

1.3.1 Part (a)

Fit a linear model using each of the features, then make predictions on the test data.

```
In [30]: # Separate features and targets in both test/train data

         # Training features, target
         y_train = train_df.rentals
         X_train = train_df.drop(['rentals'], axis=1)

         # Test features, target
         y_test = test_df.rentals
         X_test = test_df.drop(['rentals'], axis=1)
```

```
In [31]: %%capture
         # import sklearn linear regression modules
         import sklearn
         from sklearn.linear_model import LinearRegression
         from sklearn.metrics import mean_squared_error

         # Fit the linear model
         lm = LinearRegression()
         lm.fit(X_train, y_train)

         # Make the prediction, alter by 1.6 for increased growth
         altered_pred_y = 1.6*lm.predict(X_test)

         # Get mean squared error
         mse = mean_squared_error(altered_pred_y,y_test)
         print("MSE: ", mse)
```

1.3.2 Part (b)

Fit the additive model in pygam with LinearGAM, make altered predictions, and compare the MSE's.

```
In [32]: %%capture
         from pygam import LinearGAM

         gam = LinearGAM().gridsearch(X_train,y_train,lam = np.logspace(-5, 3, 30))
```

```
In [33]: %%capture
         cv_lam = gam.lam
         gam = LinearGAM(lam=cv_lam).fit(X_train,y_train)
```

```
In [34]: y_pred_gam = 1.6*gam.predict(X_test)
```

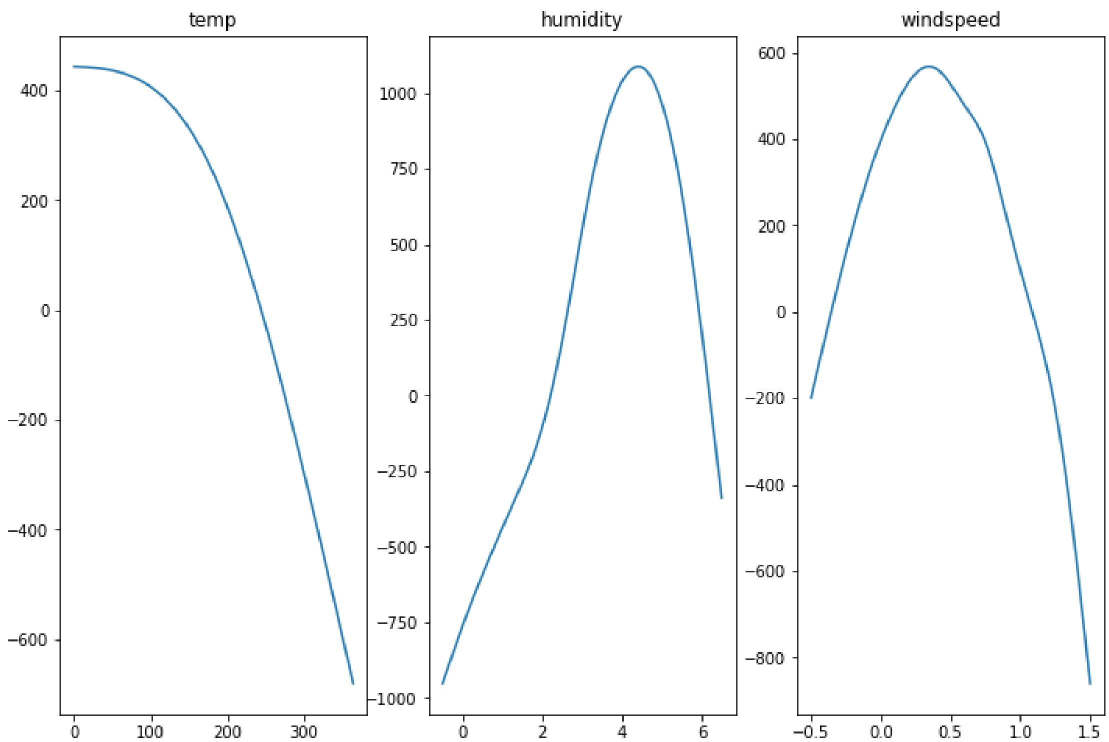
```
In [35]: mse_gam = mean_squared_error(y_pred_gam, y_test)
         print("MSE using GAM: ", mse_gam)
```

```
MSE using GAM:  962170.271987
```

1.3.3 Part (c)

Plotting the function for each of the smooth covariates (i.e., temp, humidity, and windspeed, since all the others are binary).

```
In [36]: from pygam.utils import generate_X_grid

         XX = generate_X_grid(gam)

         fig, axs = plt.subplots(1, 3)
         titles = train_df.columns.tolist()

         for i, ax in enumerate(axs):
             pdep, confi = gam.partial_dependence(XX, feature=i+4, width=.95)
             ax.plot(XX[:, i], pdep)
             ax.set_title(titles[i+4])

         plt.show()
```



In each of the cases above, the functions are clearly non-linear. *Humidity* and *Windspeed* have similar charactersitics, in that the seem somewhat like the function $f(x) = -x^2$, whereas *Temp* seems to be a downward exponential. In any case, features do not exhibit linear behavior.

1.3.4 Part (d)

Let's use a new (idealized) loss function. The function below was provided, but we examine the linear model and the spline now to compare the errors.

In [37]:
```
# Function provided by Prof. G'sell
def real_loss(true, guess):
    return ( np.mean(np.where(true>guess, 5*(true-guess), guess-true)) )


print("Linear real loss: ", real_loss(y_test, altered_pred_y))
print("Spline real loss: ", real_loss(y_test, y_pred_gam))
```

```
Linear real loss:  2412.83028509
Spline real loss:  2066.26127246
```

From the above, we observe that we'd prefer to use the additive model.

1.3.5 Part (e)

Moving away from averages due to asymmetric loss functions.

```
In [38]: # Define the vector of residuals
         resid_gam = y_test - y_pred_gam
         resid_lin = y_test - altered_pred_y

         # Obtain the quantiles
         gam_q = np.percentile(resid_gam, 500/6.0)
         lin_q = np.percentile(resid_lin, 500/6.0)

         print("gam_Q: ", gam_q)
         print("lin_Q: ", lin_q)

         # Update the estimates by the appropriate quantiles
         y_pred_gam     += gam_q
         altered_pred_y += lin_q

         # Obtain the real loss for both, relative to target y_test
         print("Linear adjusted real loss: ", real_loss(y_test, altered_pred_y))
         print("Spline adjusted real loss: ", real_loss(y_test, y_pred_gam))
```

```
gam_Q:  825.003299274
lin_Q:  901.242818149
Linear adjusted real loss:  1429.4534003
Spline adjusted real loss:  1348.85305454
```

1.4 Question 4

1.4.1 Part (a)

If we force $b_0(x) = 1 \forall x$, we may write,

$$f(x) = \sum_{k=0}^{K} \beta_i b_k(x)$$

This cleans up the algebra a little bit. Define the matrix $\mathbf{B} \in \mathbb{R}^{n \times (K+1)}$ as follows,

$$\mathbf{B} = \begin{bmatrix} 1 & b_1(x_1) & \dots & b_K(x_1) \\ 1 & b_1(x_2) & \dots & b_K(x_2) \\ \vdots & \vdots & \vdots & \vdots \\ 1 & b_1(x_K) & \dots & b_K(x_K) \end{bmatrix}$$

Also, define the the coefficient/target vector, $\vec{\beta} \in \mathbb{R}^{(K+1)\times 1}, \vec{y} \in \mathbb{R}^{n \times 1}$ as usual,

$$\vec{\beta} = \Big(\beta_0, \beta_1, \dots, \beta_K\Big)^T \vec{y} = \Big(y_0, y_1, \dots, y_K\Big)^T$$

Performing the matrix vector multiplication we see that

$$\sum_{i=1}^{n}\left(y_i - f(x_i)\right)^2 = \left(\vec{y} - \mathbf{B}\vec{\beta}\right)^T\left(\vec{y} - \mathbf{B}\vec{\beta}\right) = ||\vec{y} - \mathbf{B}\vec{\beta}||_2^2$$

1.4.2 Part (b)

In a similar fashion, we may write $\int \left(f''(x)\right)^2 dx$ using matrix vector notation.

Firstly, since we take the derivatives the intercept term vanishes, so remove the 0^{th} component of the β vector defined in part (a) above so that $\vec{\beta} \in \mathbb{R}^{K\times 1}$.

Secondly, define the matrix $\mathbf{Q}$ as follows,

$$\mathbf{Q} = \begin{bmatrix} \int b_1''(x)b_1''(x) & \int b_1''(x)b_2''(x) & \dots & \int b_1''(x)b_K''(x) \\ \int b_2''(x)b_1''(x) & \int b_2''(x)b_2''(x) & \dots & \int b_2''(x)b_K''(x) \\ \vdots & \vdots & \vdots & \\ \int b_K''(x)b_1''(x) & \int b_K''(x)b_2''(x) & \dots & \int b_K''(x)b_K''(x) \end{bmatrix}$$

Note:: I have removed the dx terms in each of the matrix element integrals for convenience.

Performing the algebra, one finds that

$$\int \left(f''(x)\right)^2 dx = \vec{\beta}^T\mathbf{Q}\vec{\beta}$$

1.4.3 Part (c)

In the above I had changed the $\vec{\beta}$ just to make things easy in each of the cases - because I didn't realize at the time I'd be putting them together. Assume that the dimensionality of the $\vec{\beta}$ is sensible for the following derivation, as it's just cumbersome to rewrite everything solely for the intercept term when the idea remains clear .

We would like to solve the minimization problem,

$$\arg\min_f \sum_{i=1}^{n}\left(y_i - f(x_i)\right)^2 + \lambda\int \left(f''(x)\right)^2 dx$$

We may rewrite the problem using the results from part(a) and part(b) as,

$$\arg\min_f ||y - \mathbf{B}\beta||_2^2 + \lambda\beta^T\mathbf{Q}\beta$$

Note: I'm omitting th vector notation since it looks terrible in Latex, any advice on this?

To be truthful, I had to look up how to do matrix differentiation because I had no idea, but I believe the following is the correct idea as it leads to an equation for β that resembles ridge regression.

Differentiating wrt β in the above equation and setting equal to zero, we arrive at

$$0 = -2\mathbf{B}^Ty + 2\mathbf{B}^T\mathbf{B}\hat{\beta} + 2\lambda\mathbf{Q}\hat{\beta}$$

We solve for $\hat{\beta}$ and find that,

$$\hat{\beta} = \left(\mathbf{B}^T\mathbf{B} + \lambda\mathbf{Q}\right)^{-1}\mathbf{B}^Ty$$

This looks very similar to ridge regression, except a few things. First, I notice that our $\mathbf{B}$ is in replace of the usual observation matrix $\mathbf{X}$ - this is sensible since we have chosen a new basis $\{b_k(x)\}_{k=0}^K$ to define the function. The second difference is rather than the identity as in ridge regression, we have $\lambda\mathbf{Q}$. Since before our regularization was on the coefficient $||\beta||_2^2$, but it is now on $\int \left(f''(x)\right)^2$, this is sensible as well becauase we may write $||\beta||_2^2 = \beta^T I \beta$ but we have replaced I with $\mathbf{Q}$ and write $\int \left(f''(x)\right)^2 = \beta^T \mathbf{Q} \beta$.