Homework 2_zx1137_DS-GA 1013

February 20, 2021

Name: Zhuoyuan Xu (Kallen Xu)

NetID: zx1137

Due Date: Feb 21, 2021

Problem 1

The idea behind PCA can be described as minimizing the reconstruction error, i.e. the equared distance between the original data and its estimate. Given a dataset of n p-dimensional data points $x_1, ..., x_n \in R^p$, we can define the reconstruction of data in R^q to R^p as

$$f(\lambda) = \mu + v_q\lambda$$

where the mean is $\mu \in R^p$, v_q is a $p \times q$ matrix with q orthogonal unit vector, $\lambda \in R^q$ is the low-dimensional data points. Then minimizing the reconstruction error can be written as

$$\min_{\mu, \lambda_{1...N}, v_q} \sum_{n=1}^{N} ||x_n - (\mu + v_q\lambda_n)||$$

Assume the data is centered, this equation can be further written as

$$\min_{v_q} \sum_{n=1}^{N} ||x_n - v_q v_q^T x_n||^2$$

where v_q are the principle components. Consider SVD to get the solution of v_q, $\Sigma_{\tilde{x}} = UDV^T$

This shows that PCA geometrically minimizes the distance of the data projecting onto the principle direction, or the length of vertical line segment from the point to the direction.

In a least square problem, we aim to minimize the least squares objective function

$$\min_{\beta} ||X\beta - y||^2$$

In this problem, we set $x_i[1]$ to be the feature, and $x_i[2]$ the corresponding response, so the formula can be written as

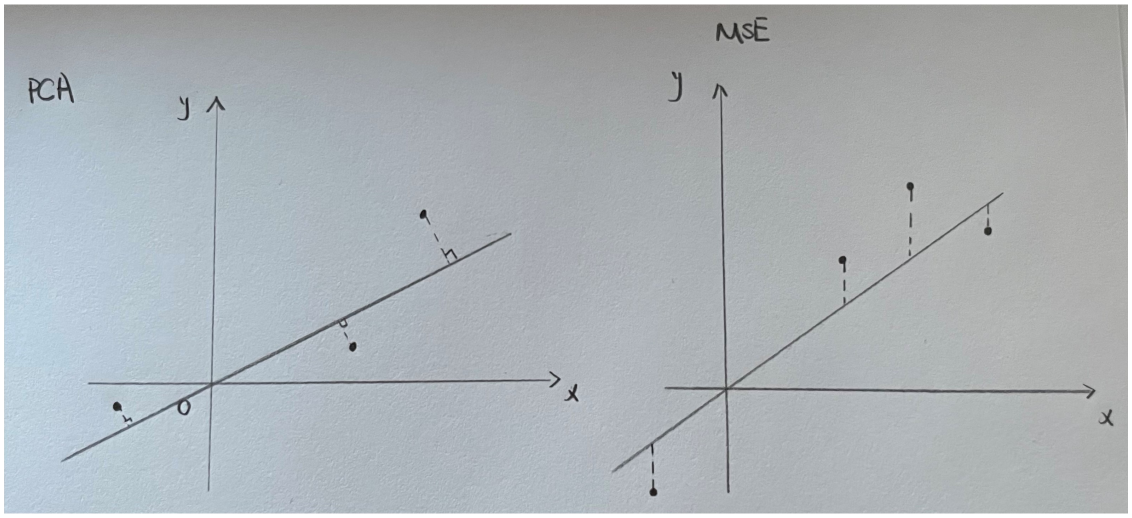
$$\min_{\beta} \sum_{i=1}^{N} ||x_i[1]\beta - x_i[2]||^2$$

The formula shows MSE aims to minimize the differences between the y-value or response of the estimated points and the observed response, or the length of the line connecting the estimated point and the observed point given the same x.

The differenct can be illustrated by the following figure.

[1]:
```
from IPython.display import Image
Image(filename='Problem1.jpg')
```

[1]:



If we generate some random points, we can still see the results of PCA and MSE have slight differences.

[23]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
```

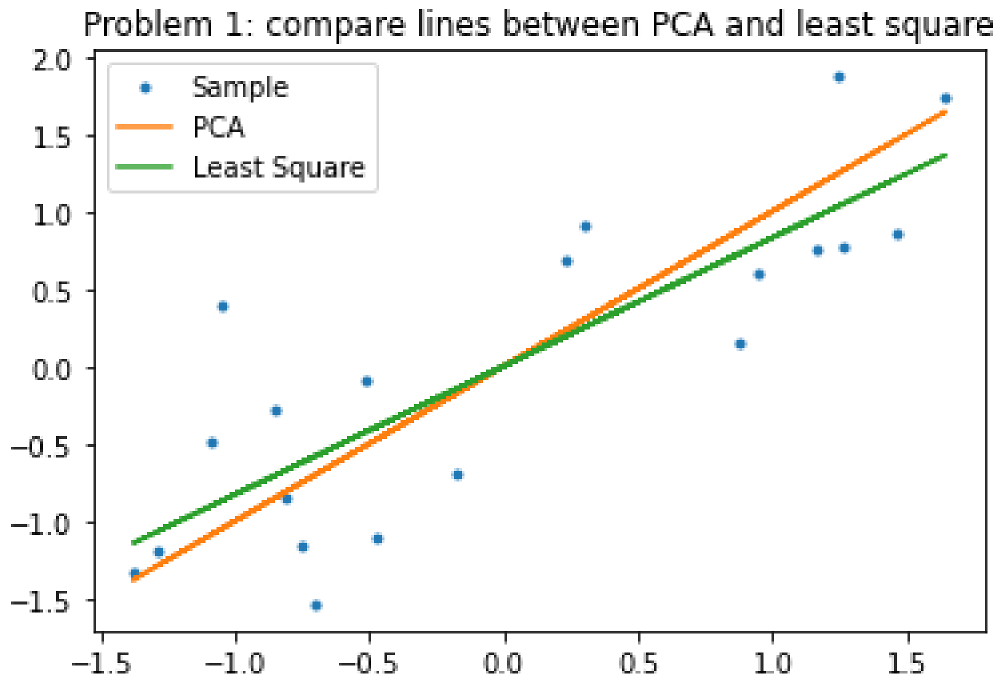
[24]:
```
# generate some random points as example to illustrate
N = 20 #number of random 2D points
X = np.random.rand(N, 1)
X = StandardScaler().fit_transform(X) #x_i[1]

# the true responses x_i[2] are set as a line with some random noises
y = np.add(X, 5 * np.random.rand(N, 1))
y = StandardScaler().fit_transform(y)
```

[25]:
```
pca = PCA(n_components=1)
Y_pca = pca.fit_transform(X)
```

[26]:
```
reg = LinearRegression().fit(X, y)
Y_reg = reg.predict(X)
```

[27]:
```
plt.plot(X, y, '.', label = 'Sample')
plt.plot(X, Y_pca, label = 'PCA')
plt.plot(X, Y_reg, label = 'Least Square')
plt.title('Problem 1: compare lines between PCA and least square')
plt.legend()
```

```
[27]: <matplotlib.legend.Legend at 0x16bf97ae730>
```



Problem 2

(a) To compute the best linear estimate of $\tilde{b}$ given $\tilde{x}[1]$ in terms of MSE, the estimator can be written as

$$\beta^* = \Sigma_{\tilde{x}[1]}^{-1}\Sigma_{\tilde{x}[1]\tilde{b}}$$

Given the variables are uncorrelated with each other,

$$\begin{aligned}\Sigma_{\tilde{x}[1]} =& E(\tilde{x}[1]^2)\\ =& E(\tilde{b}^2) + E(\tilde{m}^2) + E(\tilde{z_1}^2)\\ =& 1 + 10 + 1\\ =& 12\end{aligned}$$

$$\begin{aligned}\Sigma_{\tilde{x}[1]\tilde{b}} =& E(\tilde{x}[1]\tilde{b})\\ =& E((\tilde{b} + \tilde{m} + \tilde{z}_1)\tilde{b})\\ =& E(\tilde{b}^2) + E(\tilde{b}\tilde{m}) + E(\tilde{z}_1\tilde{b})\\ =& 1 + 0 + 0\\ =& 1\end{aligned}$$

Thus,

$$\beta^* = \frac{1}{12}$$

$$\hat{b}(\tilde{x}[1]) = \tilde{x}[1]\beta^* = \frac{1}{12}\tilde{x}[1]$$

The corresponding MSE can be written as

$$E((\tilde{b} - \tilde{x}[1]^T \Sigma_{\tilde{x}[1]}^{-1} \Sigma_{\tilde{x}[1]\tilde{b}})^2) = Var(\tilde{b}) - \Sigma_{\tilde{x}[1]\tilde{b}}^T \Sigma_{\tilde{x}[1]}^{-1} \Sigma_{\tilde{x}[1]\tilde{b}} = 1 - \frac{1}{12} = \frac{11}{12}$$

(b) To compute the best linear estimate of $\tilde{b}$ given $\tilde{x}$ in terms of MSE, we can first compute the variance and covariance of some given variables.

$$\begin{aligned} E(\tilde{x}[1]^2) &= 12 \\ E(\tilde{x}[2]^2) &= E(\tilde{m}^2) + E(\tilde{z_2}^2) \\ &= 10 + 1 \\ &= 11 \\ E(\tilde{x}[1]\tilde{x}[2]) &= E(\tilde{m}^2) \\ &= 10 \\ E(\tilde{x}[1]\tilde{b}) &= 1 \\ E(\tilde{x}[2]\tilde{b}) &= 0 \end{aligned}$$

$$\Sigma_{\tilde{x}\tilde{b}} = \begin{bmatrix} E(\tilde{x}[1]\tilde{b}) \\ E(\tilde{x}[2]\tilde{b}) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \Sigma_{\tilde{x}} = \begin{bmatrix} 12 & 10 \\ 10 & 11 \end{bmatrix}$$

$$\beta^* = \Sigma_{\tilde{x}}^{-1} \Sigma_{\tilde{x}\tilde{b}} = \frac{1}{132 - 100} \begin{bmatrix} 11 & -10 \\ -10 & 12 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{32} \begin{bmatrix} 11 \\ -10 \end{bmatrix}$$

Thus,

$$\hat{b}(\tilde{x}) = \tilde{x}^T \beta^* = \frac{1}{32} \tilde{x}^T \begin{bmatrix} 11 \\ -10 \end{bmatrix} = \frac{1}{32}(11\tilde{x}[1] - 10\tilde{x}[2])$$

The corresponding MSE can be written as,

$$E((\tilde{b} - \tilde{x}^T \Sigma_{\tilde{x}}^{-1} \Sigma_{\tilde{x}\tilde{b}})^2) = Var(\tilde{b}) - \Sigma_{\tilde{x}\tilde{b}}^T \Sigma_{\tilde{x}}^{-1} \Sigma_{\tilde{x}\tilde{b}} = 1 - \frac{1}{32} \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 11 & -10 \\ -10 & 12 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{21}{32}$$

Problem 3

(a) Given the linear regression model

$$\tilde{y} = X^T\beta + \tilde{z}$$

we can compute expectation as

$$E(C\tilde{y}) = CE(\tilde{y}) = C(E(X^T\beta) + E(\tilde{z})) = CX^TE(\beta) + E(C\tilde{z})$$

Since $\tilde{z}$ is assumed to have mean 0 and random, if we can have

$$E(C\tilde{z}) = 0$$

which means $\tilde{z}$ is uncorrelated with $\tilde{y}$. Then,

$$E(C\tilde{y}|X^T) = CX^TE(\beta)$$

If we can have $CX^T = I$, then

$$E(C\tilde{y}) = \beta$$

and $C\tilde{y}$ is unbiased.

(b) The covariance matrix of $C\tilde{y}$ is

$$\begin{aligned}Var(C\tilde{y}|X^T) &= E((C\tilde{y}-\beta)(C\tilde{y}-\beta)^T|X^T)\\ &= E(C\tilde{z}\tilde{z}^TC^T|X^T)\\ &= C(\sigma^2 I)C^T\\ &= \sigma^2 CC^T\end{aligned}$$

(c) Since we set $D = C - (XX^T)^{-1}X$, from part (a), we can know that $CX = I$ if $C\tilde{y}$ is an unbiased estimator of β.

$$CX = DX^T + ((XX^T)^{-1}X)X^T = I$$

Thus, $DX^T = 0$

(d) The covariance of $C\tilde{y}$ can be written as

$$\begin{aligned}\Sigma_C &= Var(C\tilde{y}|X)\\ &= Var(((XX^T)^{-1}X + D)\tilde{y}|X)\end{aligned}$$

If $C\tilde{y}$ is an unbiased estimator of β, by the result of part (c), then

$$\begin{aligned}\Sigma_C &= \sigma^2((XX^T)^{-1}X + D)((XX^T)^{-1}X + D)^T\\ &= \sigma^2(XX^T)^{-1} + \sigma^2 DD^T\end{aligned}$$

Since Σ_{OLS} is defined as the covariance matrix of $(XX^T)^{-1}X\tilde{y}$,

$$\begin{aligned}\Sigma_C &= \Sigma_{OLS} + \sigma^2 DD^T\\ v^T\Sigma_C v &= v^T\Sigma_{OLS}v + \sigma^2 v^T DD^T v\end{aligned}$$

The quadratic form of $\sigma^2 v^T DD^T v \geq 0$, and thus

$$v^T\Sigma_C v \geq v^T\Sigma_{OLS}v$$

(e) Given the least squared estimator, $\hat{\beta} = (XX^T)^{-1}X\tilde{y}$

$$\begin{aligned}E(\hat{\beta}|X^T) &= E((XX^T)^{-1}X\tilde{y}|X^T)\\ &= E((XX^T)^{-1}X(X^T\beta + Z^T\omega + \tilde{z})|X^T)\\ &= \beta + E((XX^T)^{-1}XZ^T\omega|X^T) + E((XX^T)^{-1}X\tilde{z}|X^T)\end{aligned}$$

Since $\tilde{z}$ is assumed to have mean 0 and random,

$$E(\hat{\beta}|X^T) = \beta + E((XX^T)^{-1}XZ^T\omega|X^T)$$

If $\hat{\beta}$ is still unbiased for all possible ω, then $XZ^T = 0$

5

Problem 4

[28]:
```
import pandas as pd
```

```
[29]: data = pd.read_csv('t_data.csv')
      data.describe()
```

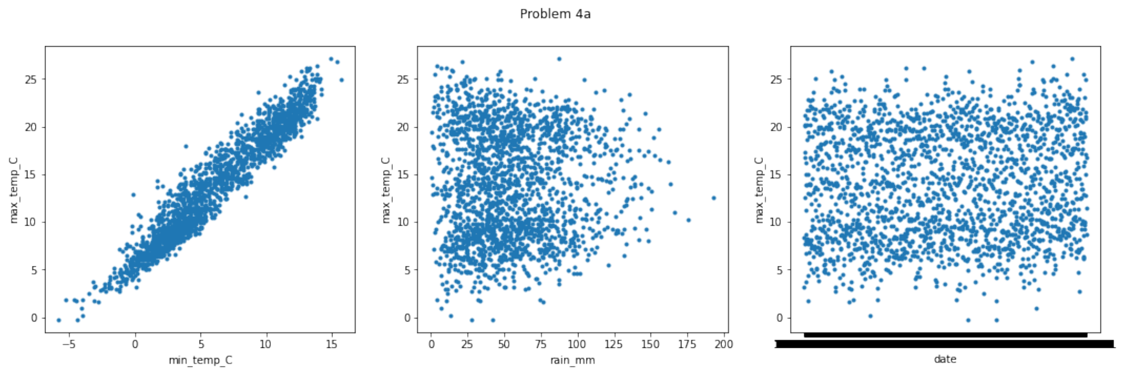
```
[29]:          max_temp_C   min_temp_C      rain_mm
      count   1944.000000  1944.000000  1944.000000
      mean      13.895833     6.181430    54.826080
      std        5.691750     4.208888    31.477289
      min       -0.200000    -5.800000     0.500000
      25%        9.000000     2.800000    31.500000
      50%       13.800000     5.600000    49.800000
      75%       19.000000    10.125000    74.900000
      max       27.100000    15.700000   192.900000
```

```
[30]: data.count()
```

```
[30]: date          1944
      max_temp_C    1944
      min_temp_C    1944
      rain_mm       1944
      dtype: int64
```

```
[31]: fig, ax = plt.subplots(1, 3, figsize = (18, 5))
      ax[0].plot(data['min_temp_C'], data['max_temp_C'], '.')
      ax[1].plot(data['rain_mm'], data['max_temp_C'], '.')
      ax[2].plot(data['date'], data['max_temp_C'], '.')
      fig.suptitle('Problem 4a')
      ax[0].set(xlabel='min_temp_C', ylabel='max_temp_C')
      ax[1].set(xlabel='rain_mm', ylabel='max_temp_C')
      ax[2].set(xlabel='date', ylabel='max_temp_C')
```

```
[31]: [Text(0, 0.5, 'max_temp_C'), Text(0.5, 0, 'date')]
```



(a) In our model, we have a, b, c, d as 4 parameters. Also, we have the hyperparameter T which we can adjust in the following problem. In our dataset, we have 150 * 12 data points as the training set. To check whether the model overfit the dataset, it is the best to check the performance of the model on the training and test set. However, generally with more data points, the risk of overfitting decreases. In this problem, since we only have 4 parameters but much more data points, we do not really have to worry about overfitting.

(b) To fit the model using least squares on the training set, we can write the equation in the form of

$$X\beta = y$$

with

$$\beta^T = \begin{bmatrix} a & b & c & d \end{bmatrix}$$

$$X = \begin{bmatrix} 1 & t_1 & cos(\frac{2\pi t_1}{T}) & sin(\frac{2\pi t_1}{T}) \\ 1 & t_2 & cos(\frac{2\pi t_2}{T}) & sin(\frac{2\pi t_2}{T}) \\ & & \dots & \\ 1 & t_n & cos(\frac{2\pi t_n}{T}) & sin(\frac{2\pi t_n}{T}) \end{bmatrix}$$

We can use both the formula and the python scipy package least square estimators that can help with calculating the coefficients at differen T.

[96]:
```python
import scipy.optimize as opt
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import normalize
```

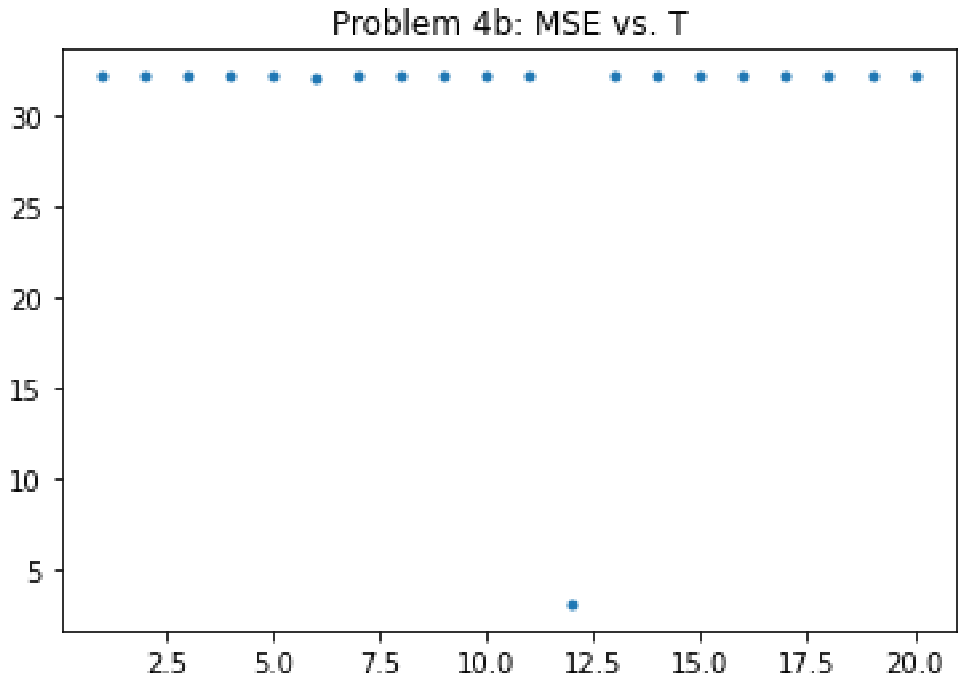
[98]:
```python
t_train = np.arange(0, 150*12, 1)
t_test = np.arange(150*12, 162*12, 1)

y_train = data['max_temp_C'][0:150*12]
y_train = y_train - np.mean(y_train)
y_test = data['max_temp_C'][150*12:]

T = np.arange(1, 21, 1)
```

[99]:
```python
def func(t, T, a, b, c, d):
    return a + b * t + c * np.cos(2*np.pi*t/T) + d * np.sin(2*np.pi*t/T)
```

[100]:
```python
# The following cell uses least square formula to calculate MSE.
MSError = []
coeff = []
for temp in T:
    X = np.empty((0, 4))
    for i in range(len(t_train)):
        X = np.append(X, np.array([[1, t_train[i], np.cos(2*np.pi*t_train[i]/
↪temp), np.sin(2*np.pi*t_train[i]/temp)]]), axis=0)
    beta = np.linalg.pinv(X.T @ X) @ X.T @ y_train.T
    coeff.append(beta)
    y_pred = func(t_train, temp, *beta)
    MSError.append(mean_squared_error(y_train, y_pred))
```

```
plt.plot(T, MSError, '.')
plt.title('Problem 4b: MSE vs. T')
```

[100]:
```
Text(0.5, 1.0, 'Problem 4b: MSE vs. T')
```



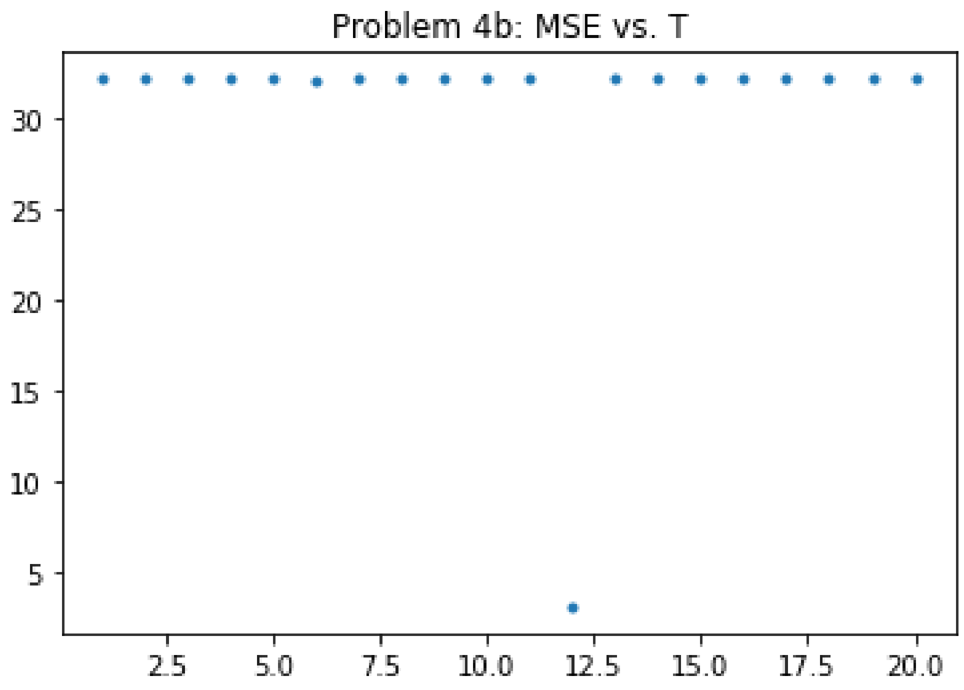
[44]:
```
# The following cell uses scipy optimize package to calculate MSE.
MSE = []
coef = []
for temp in T:
    popt1, _ = opt.curve_fit(lambda t, a, b, c, d: func(t, temp, a, b, c, d),␣
 ↪t_train, y_train)
    coef.append(popt1)
    y_pred = func(t_train, temp, *popt1)
    MSE.append(mean_squared_error(y_train, y_pred))
```

```
C:\Users\kalle\Anaconda3\lib\site-packages\scipy\optimize\minpack.py:828:
OptimizeWarning: Covariance of the parameters could not be estimated
  warnings.warn('Covariance of the parameters could not be estimated',
```

```
In this problem, we can use MSE to check which T provides the best value T^*.
```

[45]:
```
plt.plot(T, MSE, '.')
plt.title('Problem 4b: MSE vs. T')
```

[45]:
```
Text(0.5, 1.0, 'Problem 4b: MSE vs. T')
```



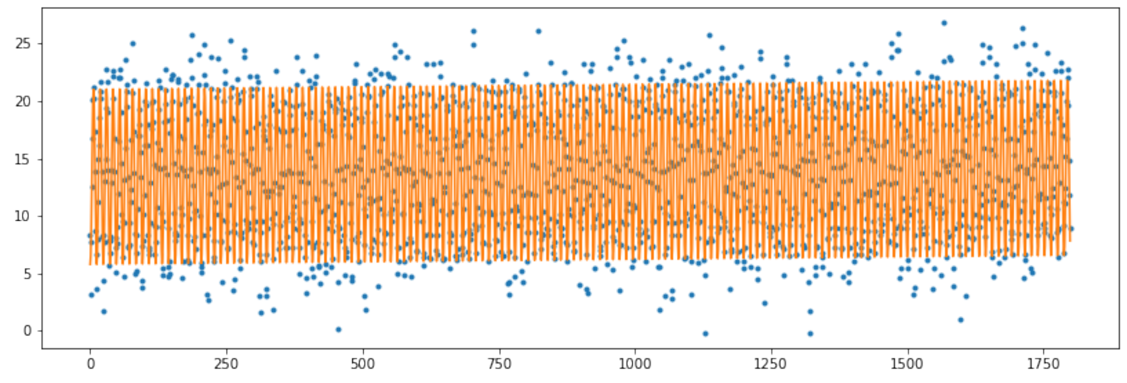
```
[37]: ind = MSE.index(np.min(MSE))
      minT = T[ind]
      print('T* with lowest MSE = ', minT)
```

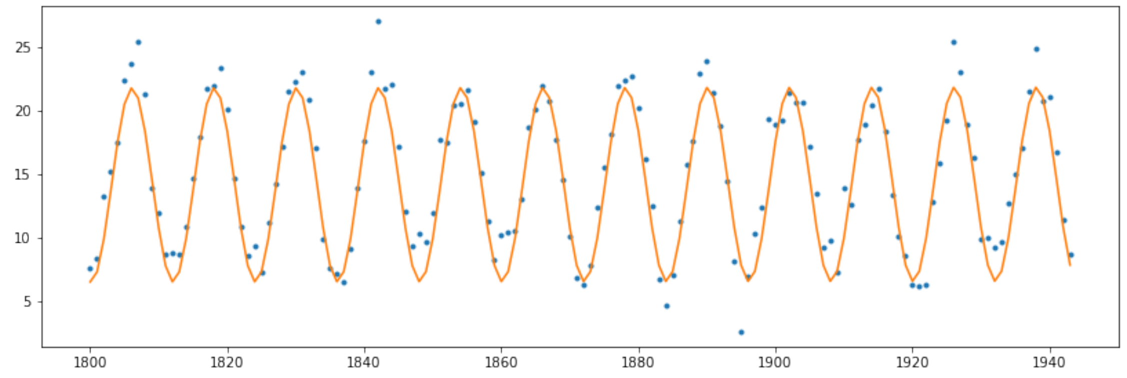
```
T* with lowest MSE =  12
```

(c)

```
[38]: fig, ax = plt.subplots(2, 1, figsize = (14, 10))
      ax[0].plot(t_train, y_train, '.')
      ax[1].plot(t_test, y_test, '.')
      ax[0].plot(t_train, func(t_train, minT, *coef[ind]))
      ax[1].plot(t_test, func(t_test, minT, *coef[ind]))
```

```
[38]: [<matplotlib.lines.Line2D at 0x16bfbc16b80>]
```





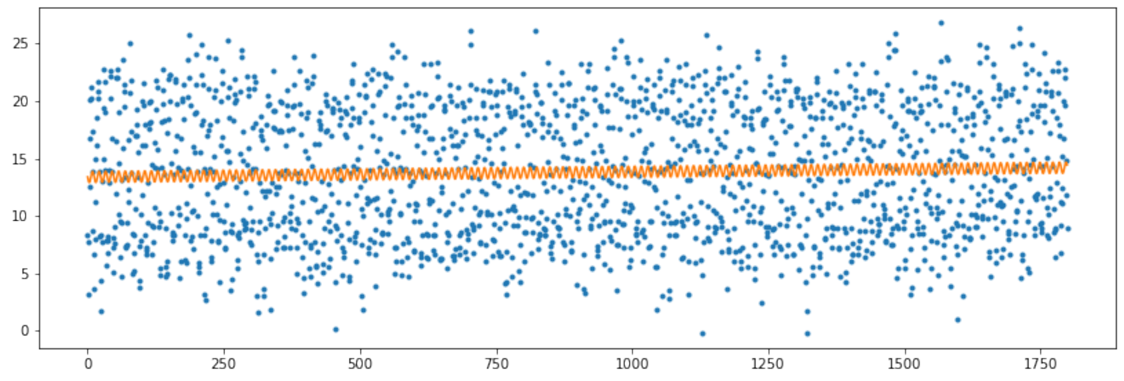
(d) Similar steps used in part (b) is applied here.

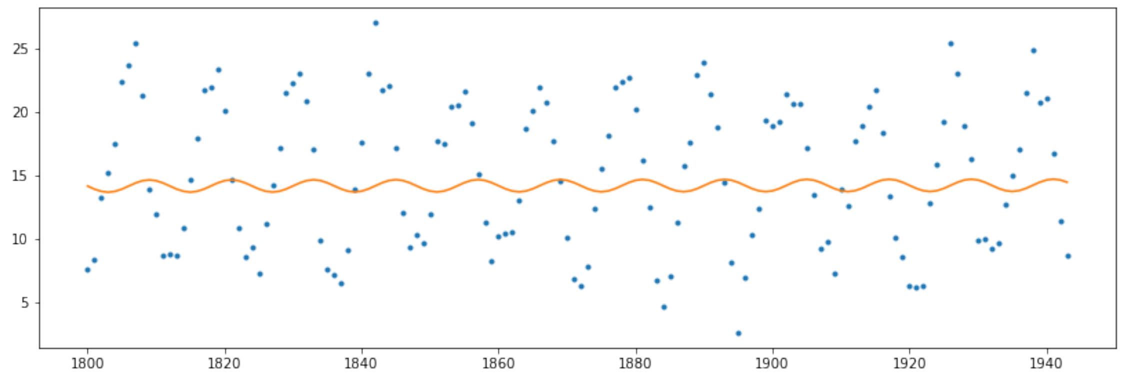
```
[39]: def func2(t, T, a, b, d):
          return a + b * t + d * np.sin(2*np.pi*t/T)
```

```
[40]: popt, _ = opt.curve_fit(lambda t, a, b, d: func2(t, minT, a, b, d), t_train,␣
       ↪y_train)
      y_pred = func2(t_test, minT, *popt)
```

```
[41]: fig, ax = plt.subplots(2, 1, figsize = (14, 10))
      ax[0].plot(t_train, y_train, '.')
      ax[1].plot(t_test, y_test, '.')
      ax[0].plot(t_train, func2(t_train, minT, *popt))
      ax[1].plot(t_test, func2(t_test, minT, *popt))
```

```
[41]: [<matplotlib.lines.Line2D at 0x16bfb75a070>]
```





The differences between the cosine and the sine function here is the shift of input or the shift of phase. The cosine function in the model shifts the input by a fixed angle to increase the possibility of getting lower error. It helps increase the flexibility of the model.

(e) In this problem, a represents the mean temperature; b is the overall trend of temperature change; c and d are the periodic fluctuations of temperature by year. If b is positive, then the overall trend of the temperature is increase, and vice versa.

```
[46]: print('The coefficients are')
      print(popt1)
```

```
The coefficients are
[1.33896667e+01 4.54202869e-04 3.53818046e-02 1.03527383e-01]
```

From the second term (b) in the above array, the temperature slightly rises in Oxford since the number is positive.

Since the temperature is periodical, we can choose temperature from the same month in different years to calculate the result. If we choose month 1 in 1853 and month 1 in 2014 for comparison, then it can be calculated by

[47]:
```
y_1853 = func2(t_train[0], minT, *popt)
y_2014 = func2(t_test[-12], minT, *popt)
print('The temperature in Oxford rises from 1853-01 to 2014-01 by ',␣
 ↪y_2014-y_1853, 'degree C.')
```

```
The temperature in Oxford rises from 1853-01 to 2014-01 by  0.8685976475053057
degree C.
```