Homework 6

1.

(a) according to the question, $X \sim exp(\lambda)$ and $s, t \geq 0$

$$Pr(X > s + t | X > s) = \frac{Pr(X > s + t, X > s)}{Pr(X > s)}$$

$$= \frac{Pr(X > s + t)}{Pr(X > s)}$$

$$= \frac{\int_{s+t}^{\infty} \lambda e^{-\lambda x} dx}{\int_{s}^{\infty} \lambda e^{-\lambda x} dx}$$

$$= \frac{-e^{-\lambda x}|_{s+t}^{\infty}}{-e^{-\lambda x}|_{s}^{\infty}}$$

$$= \frac{e^{-\lambda(s+t)}}{e^{-\lambda s}} = e^{-\lambda t}$$

Meanwhile

$$Pr(X > t) = \int_{t}^{\infty} \lambda e^{-\lambda x} dx = -e^{-\lambda x}|_{t}^{\infty} = e^{-\lambda t}$$

Therefore

$$Pr(X > s + t | X > s) = Pr(X > t)$$

which shows the memoryless property

(b)

We denote X_i the waiting time for $i\,th$ event.

Then we have $X_i \sim exp(\lambda)$, using the properties proved in the last homework, we have $X_i \sim exp(\lambda) = Gamma(1, \lambda)$ and $EX_i = \frac{1}{\lambda}$, since mean is 1 which means $EX_i = \frac{1}{\lambda} = 1 \Rightarrow \lambda = 1$

So, $X_i \sim exp(1) = Gamma(1,1)$ and with memoryless property, they are independent from each other $X_i \perp\!\!\!\perp X_j, i \neq j$

And $Z_k = \sum_{i=1}^{k} X_i \sim Gamma(k, 1)$, this is the distribution of Z_k

(c)

We let $T_1, T_2, T_3, \ldots$ represents the waiting time between consecutive events, therefore, we have $T_i \sim exp(1)$

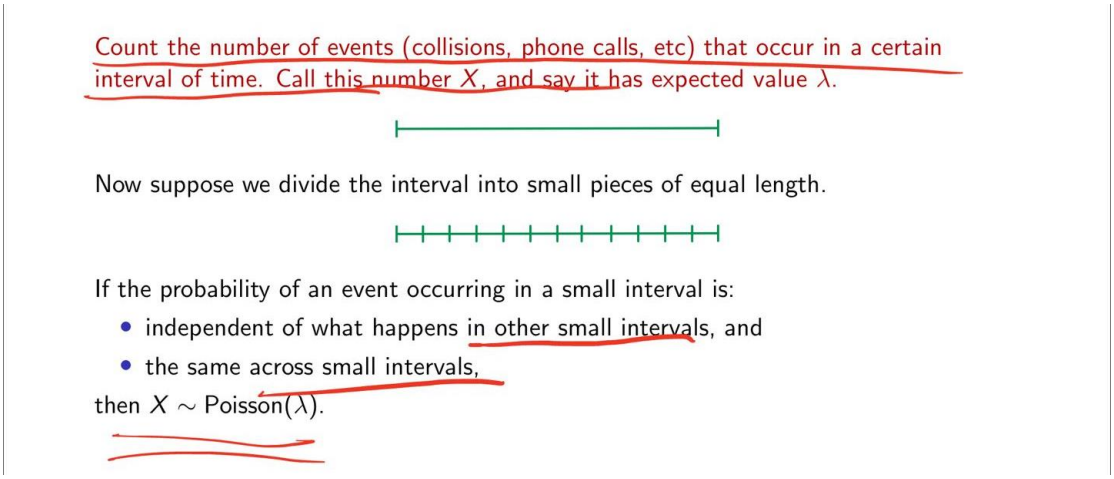
We now consider the interval time t and the number of events occurs in t. We divide it into n equal parts and n is a quite large number.

Denote Y_i as indicator variables that take the value 1 if an event occurs in the corresponding time interval, and 0 otherwise.

Then $Pr(Y_i = 0) = Pr(T_i > \frac{t}{n}) = e^{-\frac{\lambda t}{n}}$, and $Pr(Y_i = 1) = 1 - Pr(Y_i = 0) = 1 - e^{-\frac{\lambda t}{n}}$.

Now we let the number of events occurs in t be N, which N ca be represented as $N = Y_1 + Y_2 + \cdots + Y_n$.

From the definition of Poisson distribution, the probability of an event occurring in the small interval is independent of what happens in small intervals(due to memoryless property) and the same across intervals, then it should be a Poisson distribution.



According to the Poisson theorem, which is that $N \sim B(n,p)$, and $\lim_{n\to\infty} np = \lambda$, then $N \sim Poisson(\lambda)$ when n is large. Here B represents Binomial Distribution.

Then

$$\lim_{n\to\infty} np = \lim_{n\to\infty} n\,Pr(Y_i = 1)$$

$$= \lim_{n\to\infty} n\left(1 - e^{-\frac{\lambda t}{n}}\right)$$

$$= \lambda t$$

Therefore $N \sim Poisson(\lambda t)$, and its mean is $EN = \lambda t$

2.

For this question, we have

$$Y = \frac{-lnU}{\beta} \Rightarrow U = e^{-\beta Y}$$

Therefore

$$g(y) = e^{-\beta y}$$

and since $U \sim Uniform(0,1)$

$$p(u) = \begin{cases} 1, 0 \le u \le 1 \\ 0, others \end{cases}$$

Then

$$q(y) = |g'(y)| p\big(g(y)\big)$$

$$= \begin{cases} \left|-\beta e^{-\beta y}\right|, 0 \le g(y) \le 1 \\ 0, others \end{cases}$$

$$= \begin{cases} \beta e^{-\beta y}, y \ge 0 \\ 0, others \end{cases}$$

Above is distribution of Y and it's an exponential distribution

3.

(a)

We put a Dirichlet distribution on this

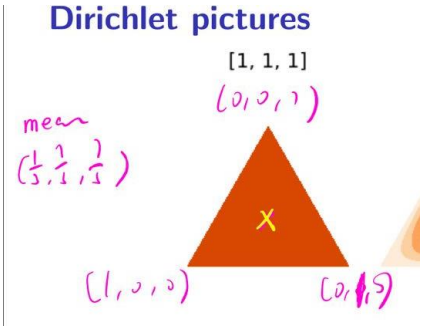
Which is

$$Pr(\theta_1, \theta_2, \theta_3) = \frac{\Gamma(\alpha_1 + \alpha_2 + \alpha_3)}{\Gamma(\alpha_1)\Gamma(\alpha_2)\Gamma(\alpha_3)} {\theta_1}^{\alpha_1 - 1} {\theta_2}^{\alpha_2 - 1} {\theta_3}^{\alpha_3 - 1}$$

over the probability simplex

$$\Delta_3 = \{(\theta_1, \theta_2, \theta_3) : \theta_i \ge 0, \sum_i \theta_i = 1, 1 \le i \le 3\}$$

Since put a uniform distribution on it, it looks like what we have been taught in the lecture shown below.



Then $\alpha_1 = \alpha_2 = \alpha_3 = 1$, and since we get count that (2,2,0), the posterior distribution will be

$$P_n(\theta) \propto P_0(\theta)\Pr(x_1, x_2, x_3|\theta)$$

$$\propto {\theta_1}^{\alpha_1 - 1}{\theta_2}^{\alpha_2 - 1}{\theta_3}^{\alpha_3 - 1}{\theta_1}^{x_1}{\theta_2}^{x_2}{\theta_3}^{x_3}$$

$$= {\theta_1}^{1-1}{\theta_2}^{1-1}{\theta_3}^{1-1}{\theta_1}^{2}{\theta_2}^{2}{\theta_3}^{0}$$

$$= {\theta_1}^{2}{\theta_2}^{2}{\theta_3}^{0}$$

Thus, $\theta \sim Dir(3,3,1)$, which has distribution function

$$Pr(\theta_1, \theta_2, \theta_3) = \frac{\Gamma(7)}{\Gamma(3)\Gamma(3)\Gamma(1)}{\theta_1}^{3-1}{\theta_2}^{3-1}{\theta_3}^{1-1}$$

$$= 180\ {\theta_1}^{2}{\theta_2}^{2}$$

(b)

Based on (a),

$$E\theta_A = \frac{\alpha_1}{\alpha_1 + \alpha_2 + \alpha_3} = \frac{3}{7}$$

$$E\theta_B = \frac{\alpha_2}{\alpha_1 + \alpha_2 + \alpha_3} = \frac{3}{7}$$

$$E\theta_C = \frac{\alpha_3}{\alpha_1 + \alpha_2 + \alpha_3} = \frac{1}{7}$$

Therefore, expected value is

$$(\theta_A, \theta_B, \theta_C) = \left(\frac{3}{7}, \frac{3}{7}, \frac{1}{7}\right)$$

4.

For this question, my design for rejection sampling is as follows

(a) Firstly, since $p(x) \propto exp\left(-\left|\left|x\right|\right|_1\right)$, p is Bivariate Exponential Distribution, and we choose our q as Bivariate Uniform Distribution over $[0,1]^2$. And each entry can be drawn from the given black box function independently, and every time we draw a pair, the pair are in the $[0,1]^2$.

(b) We choose a large number M such that M satisfies $M \geq \frac{p(x)}{q(x)}$

(c) Draw a pair x from Bivariate Uniform Distribution $q(x)$ by given black box function.

(d) Draw an u also from given black box function making $u \sim Uniform(0,1)$.

(e) If $u < \frac{p(x)}{Mq(x)}$, then output pair x and break, otherwise repeat (c)(d)

Above is the design for rejection sampling.

Similar to the lecture,

$$Pr(procedure\ that\ outputs\ a\ x)$$

$$= Pr(outputs\ x\ on\ a\ particular\ trial)$$

$$= q(x)\frac{p(x)}{Mq(x)} = \frac{p(x)}{M}$$

And

$$Pr(output\ something\ on\ a\ given\ trial)$$

$$= \sum_x Pr(outputs\ x\ on\ a\ particular\ trial)$$

$$= \sum_{x} \frac{p(x)}{M} = \frac{1}{M}$$

$$\Rightarrow E(numbers\ of\ trials\ to\ generate\ a\ sample)$$

$$= \frac{1}{Pr(output\ something\ on\ a\ given\ trial)} = M$$

5.

Based on question4 and lecture, the approach treats the $p(x)$ as uniform distribution in the unit ball, and $q(x)$ as Multivariate Uniform Distribution which can be treated as uniform distribution in the Hypercube.

Therefore, in this specific case, the algorithm chooses $M = 1$ such that $M \geq \frac{p(x)}{q(x)}$.

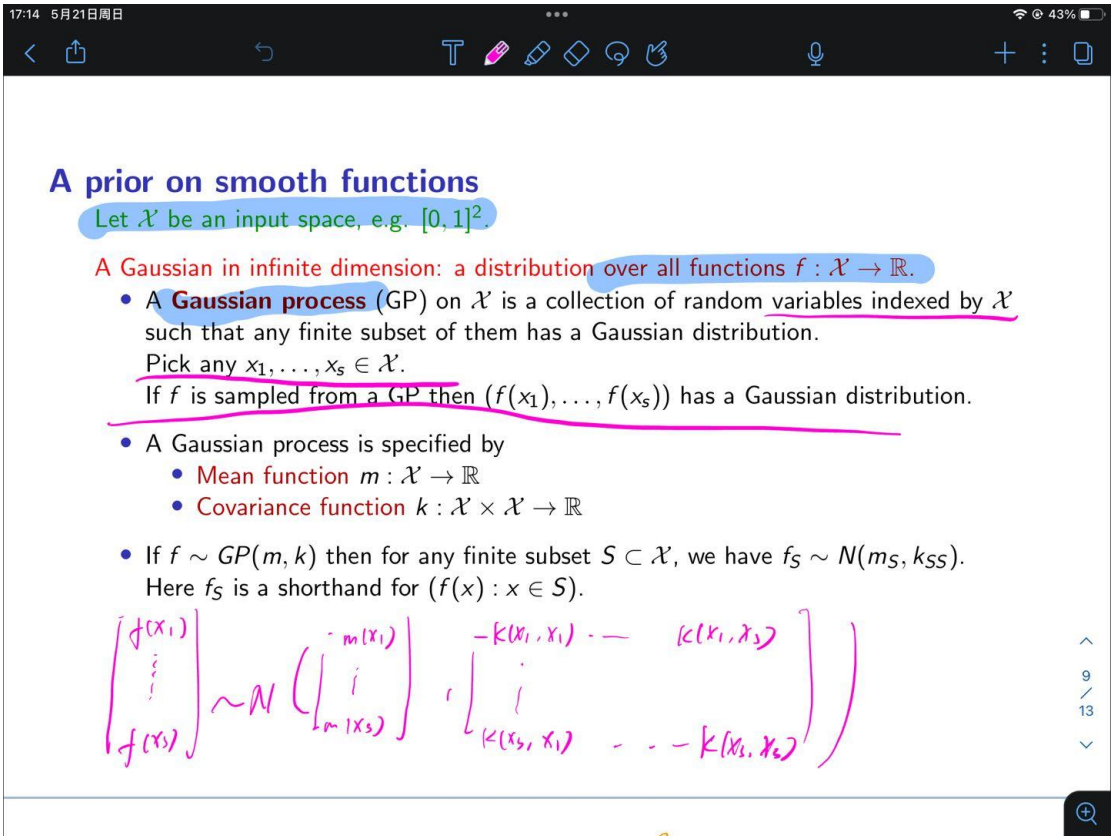
Therefore $E(numbers\ of\ trials\ to\ generate\ a\ sample) = M = 1$ for this question.

This is a good way of sampling from my perspective, M is not large, and in fact, it's the minimized one that satisfies all conditions of rejection sampling. This algorithm is similar to the Monte Carlo Methods to some extent. The expectations of numbers of generating samples is small as well which means it's quick. In the lecture, the Professor mentioned that the one disadvantage is generating is slow if M is large, and in this case, it avoids such disadvantage due to small M. All in all, I think it's a good one.

6.

(a)

The definition of Gaussian Process is as follows



Above is the definition taught in lecture.

Let $y_{tr} = [y_{tr1}, y_{tr2}, \ldots, y_{trm}]^T, y_{te} = [y_{te1}, y_{te2}, \ldots, y_{ten}]^T$ where m, n are dimensions of y_{tr} and y_{te}.

Correspondingly,

$$X_{tr} = [X_{tr1}, X_{tr2}, \ldots, X_{trm}]^T, X_{te} = [X_{te1}, X_{te2}, \ldots, X_{ten}]^T$$

$where\ X_{tri}, X_{tej}\ means\ row\ vetocr\ for\ each\ matirx\ with\ dimension\ of\ 2$

$$and\ 1 \le i \le m, 1 \le j \le n$$

And all these row vectors follow a Gaussian distribution.

Since mean function outputs 20 for every variable, then

$$m = [20,20,\dots,20]$$

Then compute covariance matrix with Covariance function $k(x,x') = exp\left(-\frac{\left||x-x'|\right|}{\sigma}\right)$

$$the\ matrix\ K$$

$$= \begin{bmatrix} k(X_{tr1},X_{tr1}) & \dots & k(X_{tr1},X_{trm}) & k(X_{tr1},X_{te1}) & \dots & k(X_{tr1},X_{ten}) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ k(X_{trm},X_{tr1}) & \dots & k(X_{trm},X_{trm}) & k(X_{trm},X_{te1}) & \dots & k(X_{trm},X_{ten}) \\ k(X_{te1},X_{tr1}) & \dots & k(X_{te1},X_{trm}) & k(X_{te1},X_{te1}) & \dots & k(X_{te1},X_{ten}) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ k(X_{ten},X_{tr1}) & \dots & k(X_{ten},X_{trm}) & k(X_{ten},X_{te1}) & \dots & k(X_{ten},X_{ten}) \end{bmatrix}$$

$$= \begin{bmatrix} K_{tr} & K_{tr,te} \\ K_{te,tr} & K_{te} \end{bmatrix}$$

According to the definition of GP, this is how we come to the conclusion.

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
#import datasets and show what's like
train=pd.read_csv('gptrain.csv',header=None)
test=pd.read_csv('gptest.csv',header=None)
test
```

Out[2]:

	0	1	2
0	-18.996684	-46.985935	16.3
1	-22.376111	-41.811944	22.1
2	-22.653579	-44.040916	20.4
3	-16.686389	-43.843889	14.8
4	-23.851944	-48.164722	19.7
5	-21.714722	-41.343889	20.5
6	-20.173333	-44.875000	18.4
7	-22.119867	-51.408637	19.1
8	-22.645833	-42.415556	22.6
9	-23.223611	-44.726944	22.9
10	-20.584444	-47.382500	17.4

In [3]:
```
#divide them into X and y and show shape
X_train,y_train=np.array(train)[:,0:2],np.array(train)[:,2:]
X_test,y_test=np.array(test)[:,0:2],np.array(test)[:,2:]
y_test.shape
```

Out[3]: (11, 1)

(b)

In [4]:
```
#write the function for computing covariance function and covariance matrix
def covariance(vec1,vec2,sigma):
    return np.exp((-1*np.linalg.norm(vec1 - vec2))/sigma)

def cov_matrix(mat1,mat2,sigma):
    m,n=mat1.shape[0],mat2.shape[0]
    mat=np.zeros((m,n))
    for i in range(m):
        for j in range(n):
            mat[i][j]=covariance(mat1[i],mat2[j],sigma)

    return mat
```

In [5]:
```
#function for compute mean
def compute_GP_mean(sigma,X_train,X_test,y_train):



    K_tr=cov_matrix(X_train,X_train,sigma)
    K_te=cov_matrix(X_test,X_test,sigma)
    K_te_tr=cov_matrix(X_test,X_train,sigma)
    K_tr_te=cov_matrix(X_train,X_test,sigma)
    print("shapes are {} , {}, {}, {}".format(K_tr.shape,K_te.shape,K_te_tr.shape,K_tr_te.shape))



    m_tr=np.array([20 for _ in range(X_train.shape[0])]).reshape(-1,1)
    m_te=np.array([20 for _ in range(X_test.shape[0])]).reshape(-1,1)

    mean_te=K_te_tr@np.linalg.inv(K_tr)@(y_train-m_tr)+m_te
    return mean_te


mean_te=compute_GP_mean(1.5,X_train,X_test,y_train)
mean_te
```

```
shapes are (93, 93) , (11, 11), (11, 93), (93, 11)
```

Out[5]:
```
array([[18.11251852],
       [21.97092198],
       [21.2117332 ],
       [15.76871024],
       [20.23993096],
       [21.2800919 ],
       [16.96205463],
       [21.84549667],
       [20.69045817],
       [20.24952044],
       [18.08471561]])
```

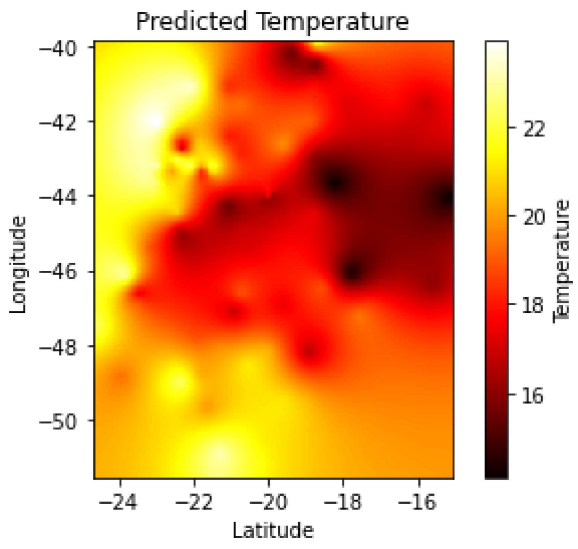
In [6]:
```
#compute mean with GP and with average predicting
mse_GP=np.mean((mean_te - y_test) ** 2)
m=np.array([np.mean(y_train) for _ in range(y_test.shape[0])])
mse_ave=np.mean((m-y_test)**2)
print("mse using gp is {}".format(mse_GP))
print("mse using ave is {}".format(mse_ave))
```

```
mse using gp is 2.4131754463183523
mse using ave is 6.422837006905684
```

(c)

In [7]:
```
import matplotlib.pyplot as plt

# Define the range of latitudes and longitudes based on the training set
min_latitude = min(X_train[:,0])
max_latitude = max(X_train[:,0])
min_longitude = min(X_train[:,1])
max_longitude = max(X_train[:,1])

# Generate a grid of test latitudes and longitudes
num_points = 75
latitudes = np.linspace(min_latitude, max_latitude, num_points)
longitudes = np.linspace(min_longitude, max_longitude, num_points)
grid_latitude,grid_longtitude = np.meshgrid(latitudes, longitudes)

grid_latitude=grid_latitude.reshape(-1,)
grid_longtitude=grid_longtitude.reshape(-1,)
grid = grid = np.column_stack((grid_latitude, grid_longtitude))


# Calculate the predicted temperatures for each point in the grid (replace with your model prediction)
predicted_temperatures = compute_GP_mean(1.5,X_train,grid,y_train)


# Reshape the predicted temperatures back into a grid
predicted = predicted_temperatures.reshape((num_points, num_points))

# Create the plot
plt.imshow(predicted, origin='lower',
           extent=[min_latitude, max_latitude, min_longitude, max_longitude], cmap='hot')
plt.colorbar(label='Temperature')  # Add a colorbar with temperature values
plt.xlabel('Latitude')
plt.ylabel('Longitude')
plt.title('Predicted Temperature')
plt.show()
```

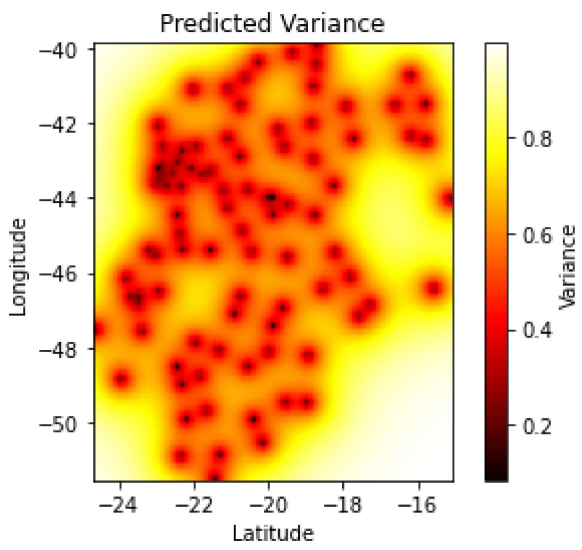
```
shapes are (93, 93) , (5625, 5625), (5625, 93), (93, 5625)
```



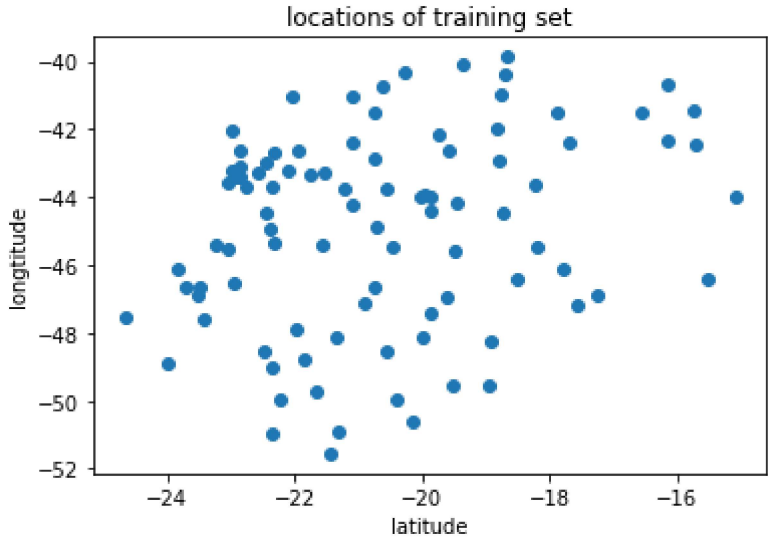
In [8]:
```
def compute_GP_variance(sigma,X_train,X_test,y_train):


    K_tr=cov_matrix(X_train,X_train,sigma)
    K_te=cov_matrix(X_test,X_test,sigma)
    K_te_tr=cov_matrix(X_test,X_train,sigma)
    K_tr_te=cov_matrix(X_train,X_test,sigma)
    print("shapes are {} , {}, {}, {}".format(K_tr.shape,K_te.shape,K_te_tr.shape,K_tr_te.shape))


    m_tr=np.array([20 for _ in range(X_train.shape[0])]).reshape(-1,1)
    m_te=np.array([20 for _ in range(X_test.shape[0])]).reshape(-1,1)

    var_te=K_te-K_te_tr@np.linalg.inv(K_tr)@K_tr_te
    return var_te
```

In [9]:
```
#Same method for the Variance
# Define the range of latitudes and longitudes based on the training set
min_latitude = min(X_train[:,0])
max_latitude = max(X_train[:,0])
min_longitude = min(X_train[:,1])
max_longitude = max(X_train[:,1])

# Generate a grid of test latitudes and longitudes
num_points = 75
latitudes = np.linspace(min_latitude, max_latitude, num_points)
longitudes = np.linspace(min_longitude, max_longitude, num_points)
grid_latitude,grid_longtitude = np.meshgrid(latitudes, longitudes)

grid_latitude=grid_latitude.reshape(-1,)
grid_longtitude=grid_longtitude.reshape(-1,)
grid = np.column_stack((grid_latitude, grid_longtitude))

predicted_variance = compute_GP_variance(1.5,X_train,grid,y_train)


diagonal = np.sqrt(np.diag(predicted_variance)).reshape((num_points, num_points))

# Create the plot
plt.imshow(diagonal, origin='lower',
           extent=[min_latitude, max_latitude, min_longitude, max_longitude], cmap='hot')
plt.colorbar(label='Variance')
plt.xlabel('Latitude')
plt.ylabel('Longitude')
plt.title('Predicted Variance')
plt.show()
```

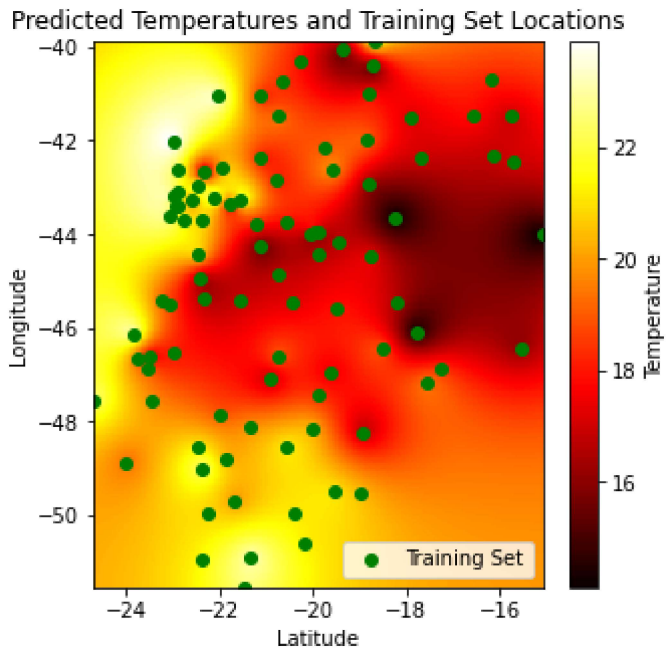
```
shapes are (93, 93) , (5625, 5625), (5625, 93), (93, 5625)
```



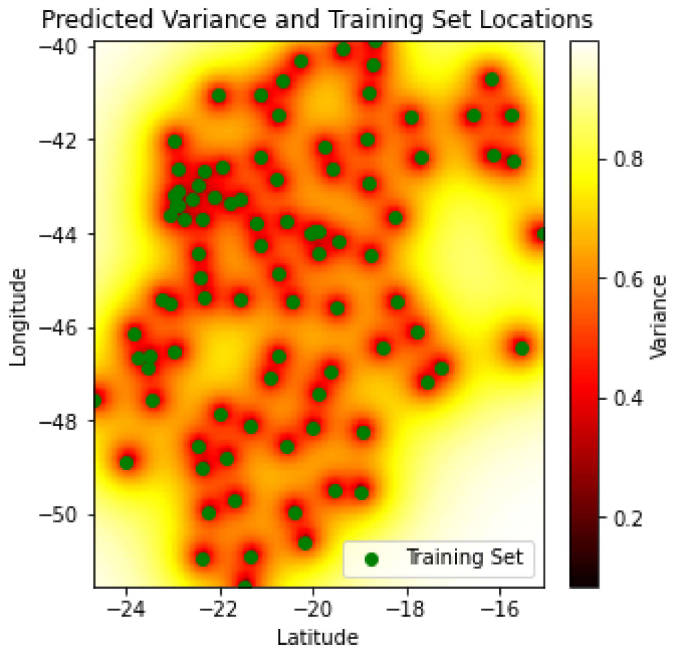
In [10]:
```
#locations of training set
fig,ax=plt.subplots()
ax.scatter(X_train[:,0],X_train[:,1])

ax.set_title("locations of training set")
ax.set_xlabel("latitude")
ax.set_ylabel("longtitude")
plt.show()
```



In [11]:
```
fig, ax = plt.subplots(1, 1, figsize=(5, 10))


im = ax.imshow(predicted, origin='lower',
               extent=[min_latitude, max_latitude, min_longitude, max_longitude], cmap='hot')
ax.set_xlabel('Latitude')
ax.set_ylabel('Longitude')
ax.set_title('Predicted Variance')
plt.colorbar(im, ax=ax, label='Temperature', shrink=0.5)


ax.scatter(X_train[:, 0], X_train[:, 1], c='green', label='Training Set')
ax.set_title('Predicted Temperatures and Training Set Locations')

ax.legend()

plt.show()
```



In [12]:
```
fig, ax = plt.subplots(1, 1, figsize=(5, 10))


im = ax.imshow(diagonal, origin='lower',
               extent=[min_latitude, max_latitude, min_longitude, max_longitude], cmap='hot')
ax.set_xlabel('Latitude')
ax.set_ylabel('Longitude')
ax.set_title('Predicted Variance')
plt.colorbar(im, ax=ax, label='Variance', shrink=0.5)

ax.scatter(X_train[:, 0], X_train[:, 1], c='green', label='Training Set')
ax.set_title('Predicted Variance and Training Set Locations')

ax.legend()

plt.show()
```



It's clear shown above that the training set locations and points with low variance are highly overlapped. It can represent that the variance is low with the short distance of that prediction to a training point.