Data Preparation:

In [2]:

```
# CS156 - Week 9.py

%matplotlib inline
%config InlineBackend.figure_format = 'svg'
import numpy as np
from scipy import optimize
import GPy
import ssgpr
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.metrics.pairwise import rbf_kernel
import datetime


print "================================================================== Train
 Data =================================================================="
years = range(2011, 2016)
files = ['CRNS0101-05-%d-CA_Yosemite_Village_12_W.txt' % y for y in years]
usecols = [1, 2, 8]

train_data = [np.loadtxt(f, usecols=usecols) for f in files]
train_data = np.vstack(train_data)

print ("Mapping from HHmm to an integer")
train_data[:, 1] = np.floor_divide(train_data[:, 1], 100) * 60 + np.mod(train_da
ta[:, 1], 100)
print ("Validating data")
valid_data = train_data[:, 2] > -1000
print ("Mapping times to day in the year:")
train_days = list(map((lambda i: datetime.datetime.strptime(str(i),
'%Y%m%d.0').timetuple().tm_yday), train_data[valid_data, 0]))
train_data_indays = zip(train_days,train_data[valid_data, 1]) #coupling a day an
d its corresponding datapoint.

x_train = np.array([(i[0] - 1) * 1440 + i[1] for i in train_data_indays])[:,np.n
ewaxis] #adding extra dimension for GPy.
y_train = train_data[valid_data, 2]
```

```
================================================================== T
rain Data ==========================================================
========
Mapping from HHmm to an integer
Validating data
Mapping times to day in the year:
```

In [3]:

```
print "================================================================== Test D
ata =================================================================="
print ("Preparing test data:")
years = [2016]
files = ['CRNS0101-05-%d-CA_Yosemite_Village_12_W.txt' % y for y in years]
usecols = [1, 2, 8]

test_data = [np.loadtxt(f, usecols=usecols) for f in files]
test_data = np.vstack(test_data)

print ("Mapping from HHmm to an integer")
test_data[:, 1] = np.floor_divide(test_data[:, 1], 100) * 60 +
np.mod(test_data[:, 1], 100)
print ("Validating data")
valid_test_data = test_data[:, 2] > -1000

print ("Mapping times to day in the year:")
test_days = list(map(lambda i: datetime.datetime.strptime(str(i), '%Y%m%d.0').ti
metuple().tm_yday,test_data[valid_test_data, 0]))
test_data_indays = zip(test_days,test_data[valid_test_data, 1])

x_test = np.array([(i[0] - 1) * 1440 + i[1] for i in test_data_indays])[:,np.new
axis] #adding extra dimension for GPy.
y_test = test_data[valid_test_data, 2]
```

```
================================================================== T
est Data ===========================================================
=======
Preparing test data:
Mapping from HHmm to an integer
Validating data
Mapping times to day in the year:
```

In [4]:

```
print "================================================================== Smalle
r Dataset =================================================================="

print "Creating a smaller dataset to test the model - comment out if you'd like
 to test on all the data:"
x_train = x_train[0::200]
y_train = y_train[0::200]

x_test = x_test[0::200]
y_test = y_test[0::200]
```

```
================================================================== S
maller Dataset =====================================================
=============
Creating a smaller dataset to test the model - comment out if you'd
 like to test on all the data:
```

Sparse Gaussian Process:

In [5]:

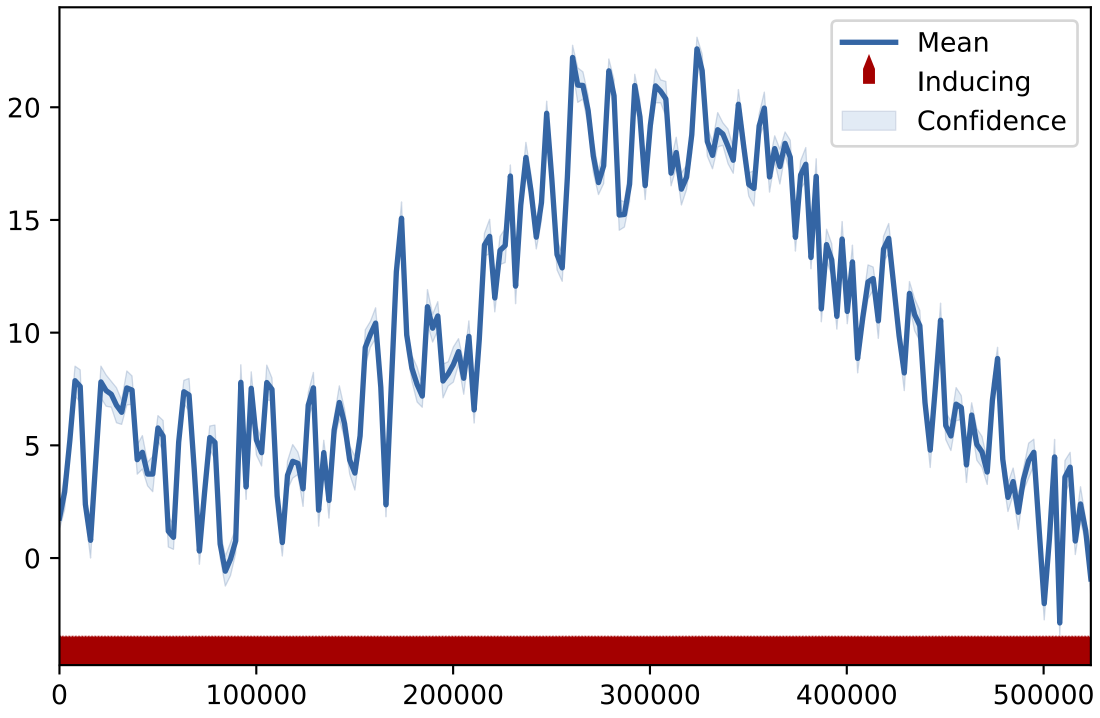
```
print "==================================================================== GPy - Model ================================================================="

"""
Describe in detail the covariance function you chose, and why. Did you fit any hyperparameters, and if so, how?
----------------------------------------------------------------------------------------------------------------
GPy used with an RBF kernel:
"""
noise_var = 0.05
#inducing points:
"""
Inducing points are picked sparsely from all across that data, with a constant step size for the SSGPR application(not used in GPy).
Those are later optimized with the model for better smoothness.
"""
inducing = 750
Z = np.linspace(0,550000,inducing)[:,None]
"""
This model from the GPy package defaults to "RBF+white", which is a variation of Radial Basis Function
with White Kernel: http://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.kernels.WhiteKernel.html.

The Kernel used is RBF, with a tweaked lengthscale.
This means that we allow the model to better fit
"""
k = GPy.kern.RBF(1,lengthscale=500)
m = GPy.models.SparseGPRegression(x_train,y_train[:,np.newaxis],kernel=k, Z=Z)

m.likelihood.variance = noise_var
m.plot(plot_data=False,plot_limits = [0,524000])

print "Optimizing the model for better smoothness - optimizing inducing points and covariance parameters now:"
m.optimize('bfgs')
m.plot(plot_data=False,plot_limits = [0,524000])
```

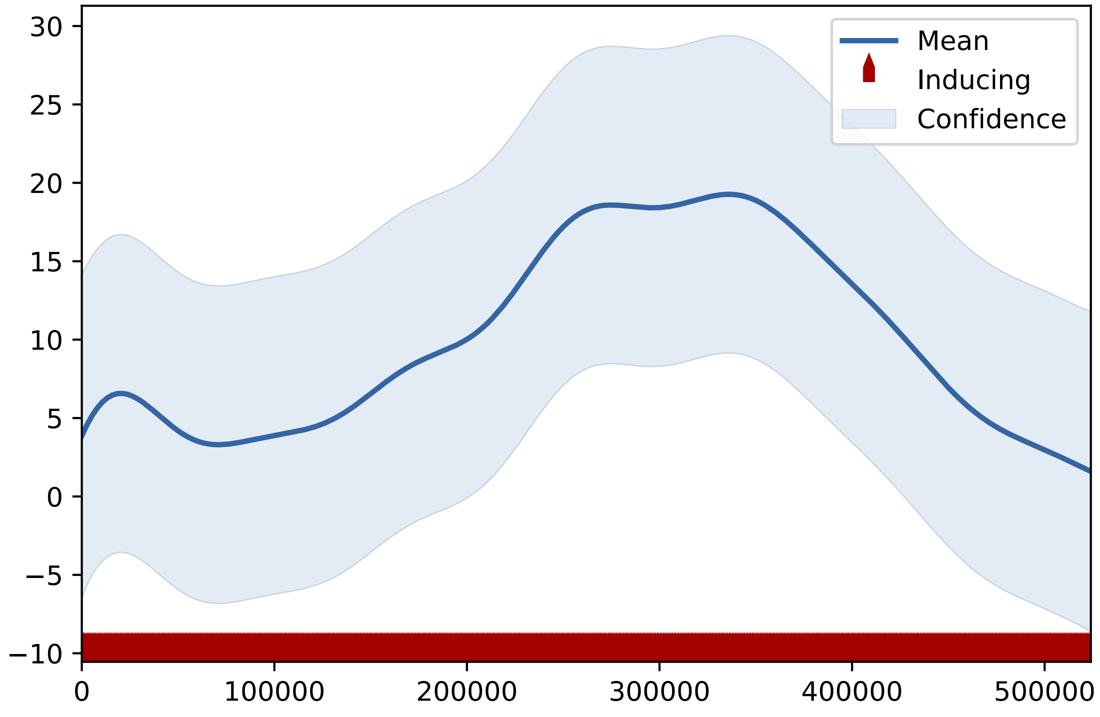
```
======================================================================
GPy - Model ==========================================================
==========
Optimizing the model for better smoothness - optimizing inducing poi
nts and covariance parameters now:
```

```
Out[5]:
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a0fe0d6d0>
```

```
 /Users/oba2311/anaconda2/lib/python2.7/site-packages/matplotlib/fig
ure.py:1999: UserWarning:This figure includes Axes that are not comp
atible with tight_layout, so results might be incorrect.
```





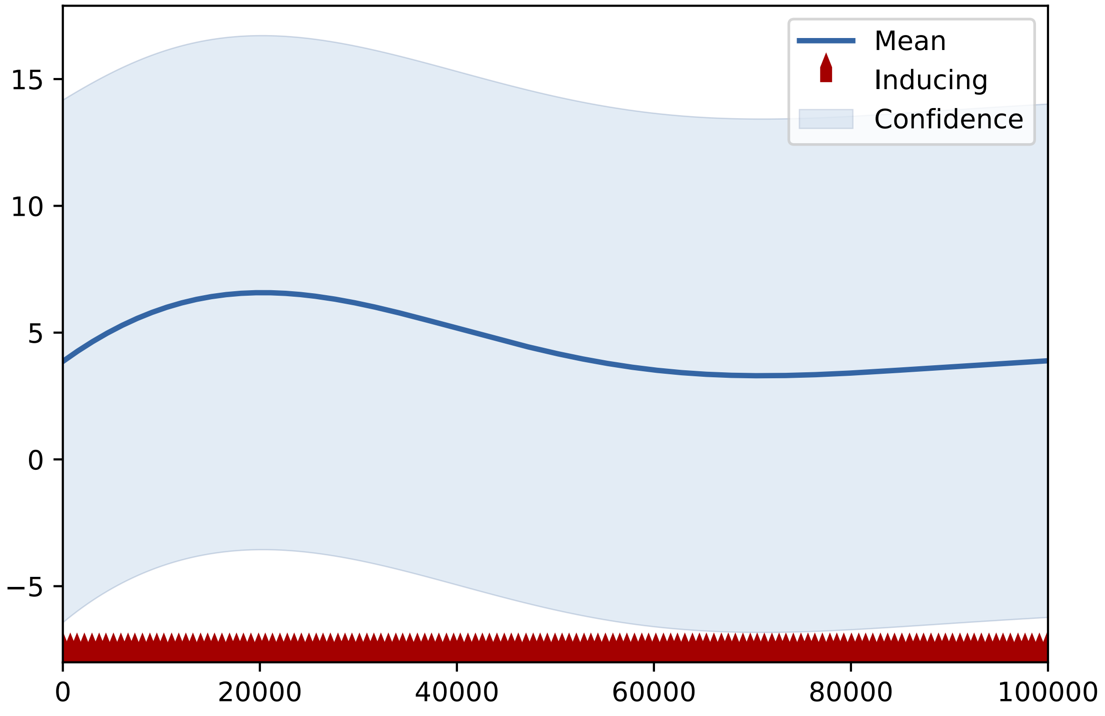
In [6]:

```
print "================================================================ Predic
tion and Scores ===============================================================
=="
#Predicting y values (temp) using the model:
m.predict(x_test)
predicted_y = m.predict(x_test)[0]
m.plot(plot_data=False,plot_limits = [0,100000])

# Measure the mean squared error (MSE) of your Gaussian process regression:
from sklearn.metrics import accuracy_score ,mean_squared_error, confusion_matrix
print "The MSE of the test set is:", mean_squared_error(y_test, predicted_y)

"""
In the assignment for week 7 the MSE was 24.7486, with a very large running tim
e.
Thus, we observe higher MSE for the Sparse GP. LPM model was quite fast to run w
hile Sprase GP took a while()
"""
```

```
================================================================ P
rediction and Scores ===============================================
===================
The MSE of the test set is: 25.5441433127
```

Out[6]:

```
'\nIn the assignment for week 7 the MSE was 24.7486, with a very lar
ge running time.\nThus, we observe higher MSE for the Sparse GP. LPM
model was quite fast to run while Sprase GP took a while()\n'
```



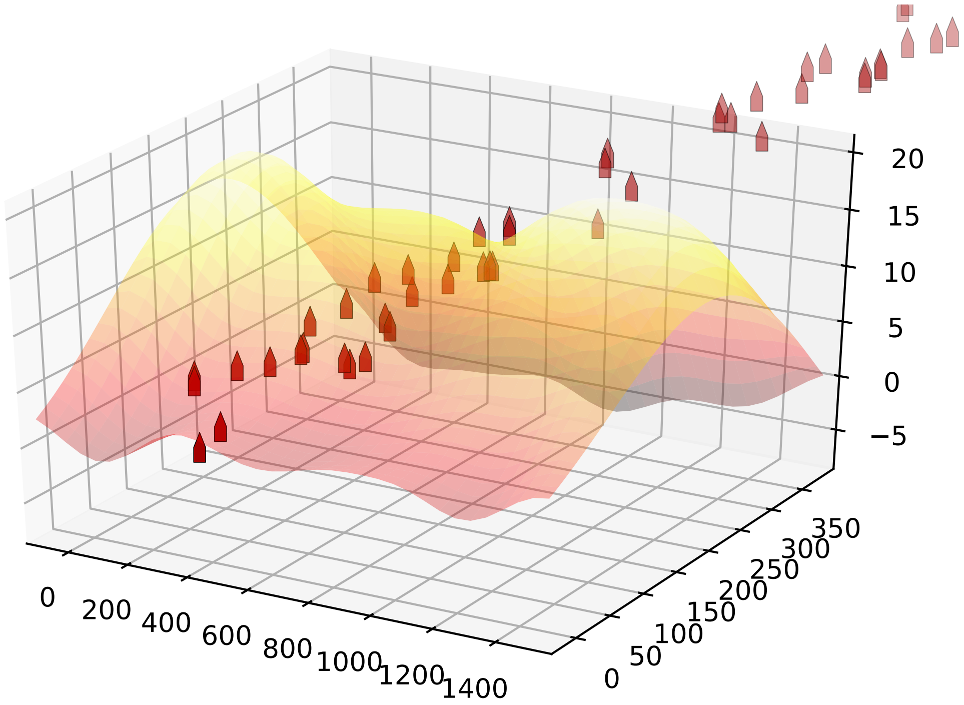
In [9]:

```
print "================================================================ 3
D plot =================================================================="


print  "plotting temperature as a function of time and day:"

k3d = GPy.kern.RBF(input_dim=2,lengthscale=200)
# y_train = np.array([y_train])
inducing = 50
# Z = np.array((np.linspace(0,365,inducing),np.linspace(0,1440,inducing)))
# Z=Z.reshape(-1,2)
# print Z
x_train = np.concatenate((np.array(train_data[valid_data, 1]).reshape(-1,1),
np.array(train_days).reshape(-1,1)),axis=1)
x_train = x_train[0::200]
y_train = np.array(y_train).reshape(-1,1)

m3d = GPy.models.SparseGPRegression(x_train,y_train,kernel=k3d,num_inducing=indu
cing)
m3d.plot(plot_data=False, projection="3d",legend=False)
```

```
====================================================================
=== 3D plot ========================================================
==========
plotting temperature as a function of time and day:
```

Out[9]:

```
<matplotlib.axes._subplots.Axes3DSubplot at 0x1a1eadf310>
```



In [ ]:

```
print "=============================================================== Stretch G
oal - SSGPR - Model ============================================================
======"

print """This probably needs to be downloaded to your computer.\nPerhaps check:
https://github.com/marcpalaci689/SSGPR \n
don't forget to make sure the .py file is in the same directory
"""

import ssgpr

x_train = np.array([(i[0] - 1) * 1440 + i[1] for i in train_data_indays])[:,np.n
ewaxis] #adding extra dimension for GPy.
y_train = train_data[valid_data, 2]

model = ssgpr.SSGPR(x_train, y_train[:,np.newaxis], inducing)
predicted_y = np.array([model])
model.plot()
```

```
=============================================================== Stre
tch Goal - SSGPR - Model ===========================================
=======================
This probably needs to be downloaded to your computer.
Perhaps check:
https://github.com/marcpalaci689/SSGPR

don't forget to make sure the .py file is in the same directory
```