In [1]:
```
import numpy as np
import pip
import pymc3 as pm
import pandas as pd
import theano as t
import theano.tensor.slinalg as tla
import theano.tensor as tt
from scipy.spatial import distance as dist
from pymc3 import NUTS, sample
from scipy import optimize
import matplotlib.pyplot as plt
```

Example with 7 points:

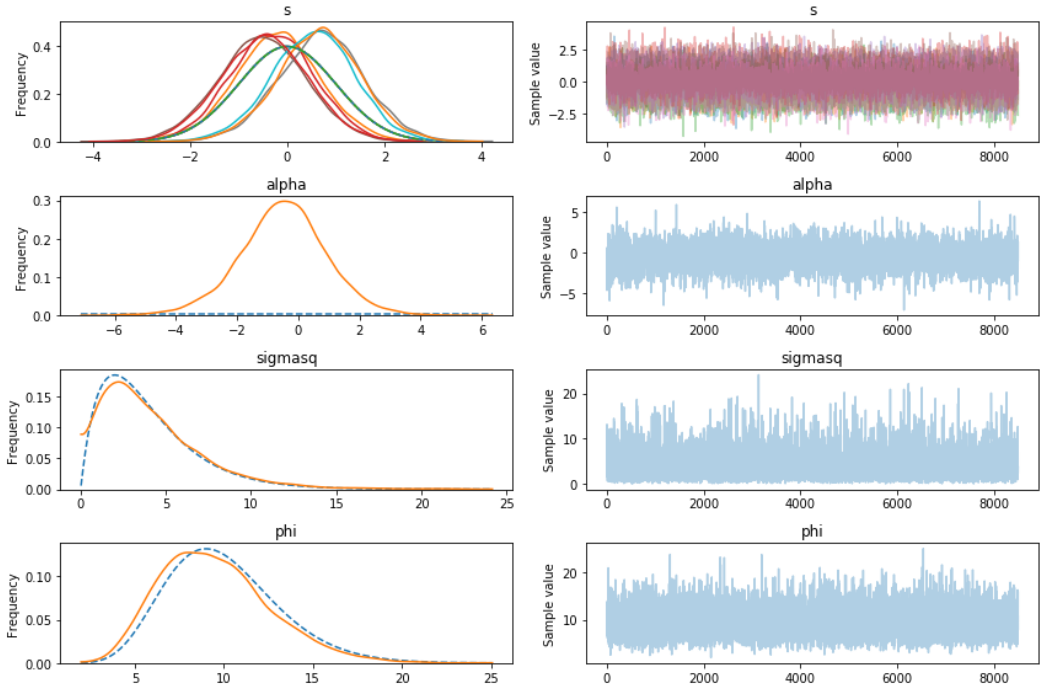
In [2]:
```
### read the data in
df_sp = pd.read_csv("7Points.csv", index_col = None)
df_sp
```

Out[2]:

	ID	success	X	Y
0	13	0	72.973	70.957
1	14	0	92.452	78.864
2	15	0	42.816	54.111
3	16	1	81.191	80.832
4	17	1	63.228	53.074
5	18	1	60.808	74.328
6	19	0	55.579	54.363

In [3]:
```
### calculate distance matrix and put it in square format
distance = dist.pdist(df_sp[['X', 'Y']], metric='euclidean')
Dist_sp = dist.squareform(distance)


### Model specification ##############################################
################################################
# Binomial runs
def singleSurface(Y, D):
    """Logistic Kriging model, where
    Y is the vector of bernoulli outcomes,
    D is a NxN matrix of distances between points"""

    W = D.shape[0]  # size of distance matrix
    I = t.shared(np.eye(W))  # identity matrix of size W
    D = t.shared(D)
    sqrt3 = tt.constant(np.sqrt(3.))

    model = pm.Model()

    with model:
        #Construct the GP
        tausq = 1e-7#pm.Gamma('tausq', 2., 1000., shape = 1, testval=0.1
)
        sigmasq = pm.Gamma('sigmasq', 2., .5, shape = 1)
        phi = pm.Gamma('phi', 10., 1., shape = 1)

        # Spatial random effect
        # Matern 3/2 covariance. The Matérn covariance functions become
        # especially simple when ν is half-integer: ν = p + 1/2, where
        # p is a non-negative integer. In this case the covariance funct
ion
        # is a product of an exponential and a polynomial of order p
        ## nu = 3/2, C(d) = sigma^2(1+sqrt(3)d/phi)*exp(-sqrt(3)d/phi)

        tmp = tt.mul(D, sqrt3 / phi) # D*sqrt(3)/phi
        Sigmasq = sigmasq * ( 1. + tmp) * tt.exp(-tmp) ## matern functio
n
        Sigmasq = Sigmasq + tausq * I
        Sigma = tla.Cholesky(lower=True)(Sigmasq)

        #Priors
        s = pm.Normal('s', mu=0., sd=1., shape=W)
        spatial_re = tt.dot(Sigma, s)
        alpha = pm.Normal('alpha', 0., 100., shape=1, testval=0.1)

        # Data model
        p = pm.invlogit(alpha + spatial_re)
        y_obs = pm.Bernoulli('y_obs', p=p, observed=Y) # logit link

        return model




# creating the model
m_sp_Matern = singleSurface(Y=np.array(df_sp[["success"]]).flatten(),
                                    D=Dist_sp)


# taking samples from the posterior
with m_sp_Matern:
    db = pm.backends.Text('Trace_7Points')   # saving trace
    trace = pm.sample(draws=4000, tune=500, n_init=100000, trace=db,
                      start={"sigmasq": 1.0, "phi": 15.0})
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
100%|██████████| 4500/4500 [00:20<00:00, 216.67it/s]
```



In [4]:
```
df_gp = pd.read_csv("Trace_7Points/chain-0.csv", index_col = None)
df_gp.head()
```

Out[4]:

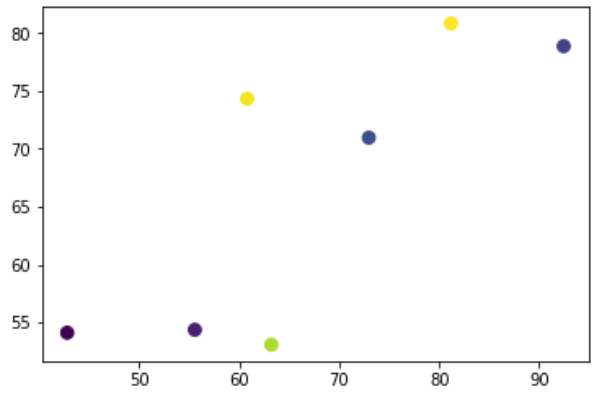
	sigmasq_log____0	phi_log____0	s__0	s__1	s__2	s__3	s__4	s_
0	1.333746	1.741777	0.562430	0.222088	-0.539378	-0.364642	0.042323	0.2948
1	1.333746	1.741777	0.562430	0.222088	-0.539378	-0.364642	0.042323	0.2948
2	1.976143	1.678069	0.590487	-0.993882	-0.116663	1.580485	0.628619	0.7558
3	1.650011	2.076347	-0.605520	-0.678313	-0.884913	0.791329	1.348509	2.0578
4	1.636399	2.272060	0.648192	-0.934318	-0.320779	1.429861	1.463746	1.1542

In [5]:
```
indices10 = [a for a in range(1000, len(df_gp), 10)]
# only get columns with s values
sdf = df_gp.iloc[indices10, 2:len(df_gp.columns)-3]
print(sdf.head())
```

```
          s__0      s__1      s__2      s__3      s__4      s__5      s__
6
1000 -0.372952 -0.146148 -0.911602  1.007600  1.798261  0.061975 -0.47977
8
1010 -0.849975  0.201454 -0.467185  1.846588  0.939448  1.487554 -0.91817
2
1020 -0.394978  0.923063 -0.841416  0.948161  1.017760  1.363039  1.05664
1
1030  1.410788 -0.432431  0.007874  1.548462  2.510067  1.165913 -0.86515
4
1040 -0.401857 -0.208866  0.261954  0.235182  0.560609  1.314413  0.51365
8
```

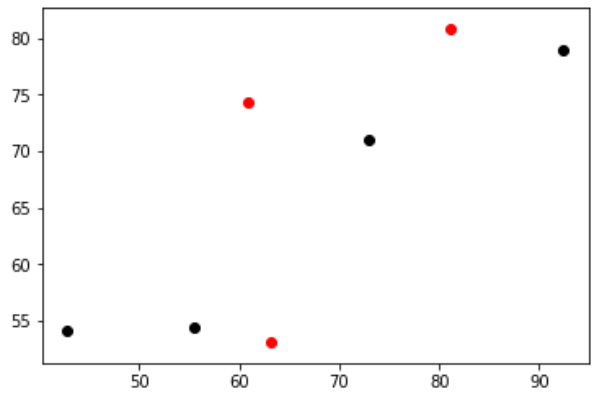
Plot Mean of each s__i column with respect to their spatial position

In [6]:
```
f, ax = plt.subplots()
x = df_sp["X"].values
y = df_sp["Y"].values
s = [60 for n in range(len(x))]
c = sdf.mean().values     # colormap: the lighter the color, the greater
the value
plt.scatter(x,y,c=c,s=s)
plt.show()
```



Original data: (red success, black failure)

In [7]:
```
def plot_points(df):
    infdf = df[df["success"] == 1]
    susdf = df[df["success"] != 1]
    plt.scatter(susdf["X"].values,susdf["Y"].values, color="k")
    plt.scatter(infdf["X"].values,infdf["Y"].values, color="r")

    plt.show()
plot_points(df_sp)
```



Works fine!

Example with 92 points:

In [8]:
```
### read the data in
df_sp = pd.read_csv("92Points.csv", index_col = None)
df_sp
```

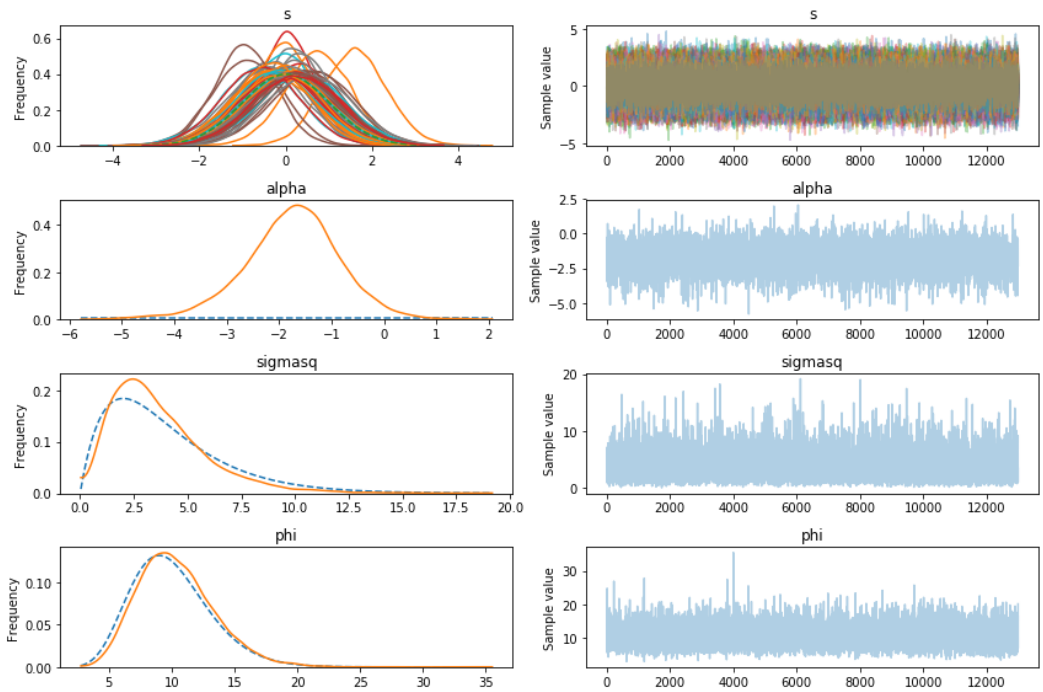
Out[8]:

	ID	success	X	Y
0	1	1	89.626	65.400
1	2	0	66.328	44.735
2	3	0	39.529	58.808
3	4	0	59.971	43.425
4	5	0	57.029	45.096
5	6	0	60.235	66.856
6	7	0	42.950	54.119
7	8	0	43.049	54.099
8	9	1	93.045	79.310
9	10	0	61.350	78.977
10	11	0	64.225	44.310
11	12	0	61.064	45.996
12	13	0	72.973	70.957
13	14	0	92.452	78.864
14	15	0	42.816	54.111
15	16	1	81.191	80.832
16	17	1	63.228	53.074
17	18	1	60.808	74.328
18	19	0	55.579	54.363
19	20	0	58.510	59.520
20	21	0	67.866	46.121
21	22	0	52.678	41.293
22	23	1	68.614	41.099
23	24	0	54.010	68.710
24	25	0	52.104	64.246
25	26	0	72.839	73.138
26	27	1	68.551	47.760
27	28	1	89.739	63.260
28	29	0	80.248	71.330
29	30	0	64.978	52.345
...	...	...	...	...
62	63	0	56.800	56.400
63	64	0	67.301	60.601
64	65	0	73.605	51.170
65	66	0	83.200	56.121
66	67	0	79.785	71.763
67	68	1	82.114	67.559
68	69	0	34.570	73.231

same inference as above

In [9]:
```
### calculate distance matrix and put it in square format
distance = dist.pdist(df_sp[['X', 'Y']], metric='euclidean')
Dist_sp = dist.squareform(distance)


# creating the model
m_sp_Matern = singleSurface(Y=np.array(df_sp[["success"]]).flatten(),
                                    D=Dist_sp)


# taking samples from the posterior
with m_sp_Matern:
    db = pm.backends.Text('Trace_92Points')   # saving trace
    trace = pm.sample(draws=4000, tune=500, n_init=100000, trace=db,
                      start={"sigmasq": 1.0, "phi": 15.0})



l=[pm.Normal.dist(mu=0., sd=1.), pm.Normal.dist(0., 100.), pm.Gamma.dist
(2.,.5, shape=1), pm.Gamma.dist(10., 1.)]
pm.traceplot(trace, priors=l)
plt.show()
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
100%|██████████| 4500/4500 [05:35<00:00, 13.40it/s]
```



In [10]:
```
df_gp = pd.read_csv("Trace_92Points/chain-0.csv", index_col = None)
df_gp.head()
```

Out[10]:

	sigmasq_log____0	phi_log____0	s__0	s__1	s__2	s__3	s__4	s
0	-0.151650	2.177401	-0.561696	-1.569584	-1.801476	0.422827	-0.603376	-0.180
1	-0.151650	2.177401	-0.561696	-1.569584	-1.801476	0.422827	-0.603376	-0.180
2	-0.395040	2.232467	-0.428991	-1.172716	-1.975827	0.293583	-0.632686	-0.035
3	1.002088	2.189286	3.674470	-0.010457	0.181263	-1.223853	0.811415	0.273
4	1.002088	2.189286	3.674470	-0.010457	0.181263	-1.223853	0.811415	0.273

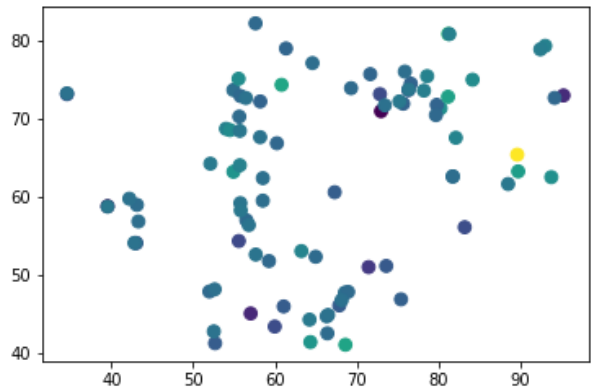
5 rows × 97 columns

In [11]:
```
# only get columns with s values
sdf = df_gp.iloc[indices10, 2:len(df_gp.columns)-3]
print(sdf.head())
```

```
          s__0      s__1      s__2      s__3      s__4      s__5      s__
6  \
1000  2.926408  0.282324  0.672303  0.433317 -0.466722 -0.459273 -2.05388
2
1010  1.312591 -0.311320 -0.987011 -0.502978 -0.355828 -0.728954 -0.74862
1
1020  1.654248  0.486238 -0.938390 -1.316920 -1.758373 -0.149189  0.59682
9
1030  2.442915  0.785504 -0.611995 -0.117601 -0.615737  0.274729 -0.47879
8
1040  1.905645  0.926247 -1.722668 -0.991794  0.078307 -1.523206 -0.75838
4

          s__7      s__8      s__9    ...        s__82     s__83     s__8
4  \
1000 -0.372213  2.479842 -0.349485    ...    -1.243777  0.071751  2.20340
0
1010 -0.443677 -0.458894 -1.250891    ...     0.855148 -0.269377 -0.69832
7
1020  1.022651 -0.089064 -0.968540    ...     2.395553 -0.606515  1.23952
5
1030 -1.385773  0.440572 -0.149378    ...    -0.521719  1.332347  1.09955
6
1040  0.167792  0.749361  0.798605    ...    -0.323306 -0.918450  1.53433
2

         s__85     s__86     s__87     s__88     s__89     s__90     s__9
1
1000  0.981438  2.335941  0.258264 -0.696459 -0.683238 -1.488770 -0.66039
2
1010 -0.657965  0.673102  1.734911  0.182539 -0.816385 -0.956995  0.72017
6
1020 -0.424076  0.682603  0.570910  0.520834  0.070787 -0.390707 -1.02445
1
1030  2.467219 -1.002151  0.565360  0.464726  2.158670  0.560592 -0.10444
2
1040  0.490538  0.408900  1.213990  0.427184  0.444551  0.674447 -0.82024
1

[5 rows x 92 columns]
```

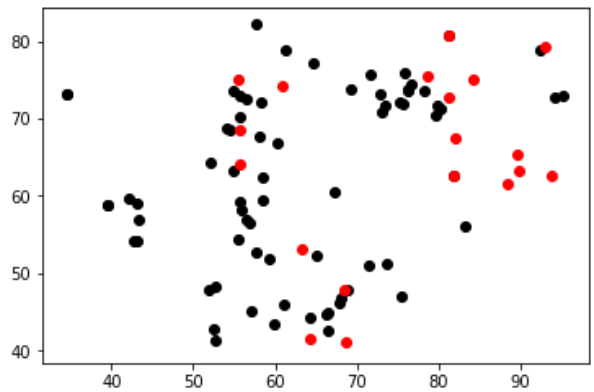
Plot Mean of each s__i column with respect to their spatial position

In [12]:
```
f, ax = plt.subplots()
x = df_sp["X"].values
y = df_sp["Y"].values
s = [60 for n in range(len(x))]
c = sdf.mean().values     # colormap: the lighter the color, the greater
the value
plt.scatter(x,y,c=c,s=s)
plt.show()
```



Original data: (red success, black failure)

In [13]:
```
def plot_points(df):
    infdf = df[df["success"] == 1]
    susdf = df[df["success"] != 1]
    plt.scatter(susdf["X"].values,susdf["Y"].values, color="k")
    plt.scatter(infdf["X"].values,infdf["Y"].values, color="r")

    plt.show()
plot_points(df_sp)
```



In [ ]:
```
```