Simulated Multivariate Kriging

June 19, 2020

Table of Contents

This notebook implements the Geostatistics Lesson . This code is provided for educational purposes and should be reviewed jointly with the lesson Collocated Cokriging.

Learning Objectives - Review simple cokriging. - Understand the why the Markov Models where developed. - Explore the differences between Markov model and Markov model . - Formulated the Kriging equations using the Markov models. - Implement the Markov model and Markov model . - Understand the Markov model and Markov model work flow(source code available).

[1]:
```
print('Package Versions:')
import matplotlib as matplotlib; print("  matplotlib:", matplotlib.__version__)
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import matplotlib.gridspec as gridspec
from mpl_toolkits.axes_grid1 import make_axes_locatable
import pandas as pd; print("  pandas:", pd.__version__)
import sys; print("  python:", sys.version_info)
import numpy as np; print("  numpy:", np.__version__)
import sklearn as sklearn; print("  sklearn:", sklearn.__version__)
import os
import scipy; print("  scipy:", scipy.__version__)
from scipy import stats
from tqdm import tqdm
from scipy.spatial import distance_matrix
from sklearn.metrics import mean_squared_error
np.set_printoptions(precision=3)
```

```
Package Versions:
  matplotlib: 2.2.2
  pandas: 0.23.0
  python: sys.version_info(major=3, minor=6, micro=10, releaselevel='final',
serial=0)
  numpy: 1.18.2
  sklearn: 0.19.1
  scipy: 1.1.0
```

1 Import Data

Consider two varable Z($\mathbf{u}$), the primary variable - Grade, and Y($\mathbf{u}$), the secondary variable - seismic. We have sample for the primary variable Z($\mathbf{u}$) at 64 locations. The secondary variable

Y($\mathbf{u}$) is measured at all location within our domain. It is considered in a probabilistic sense to use Y($\mathbf{u}$) to inform the prediction of Z($\mathbf{u}$).

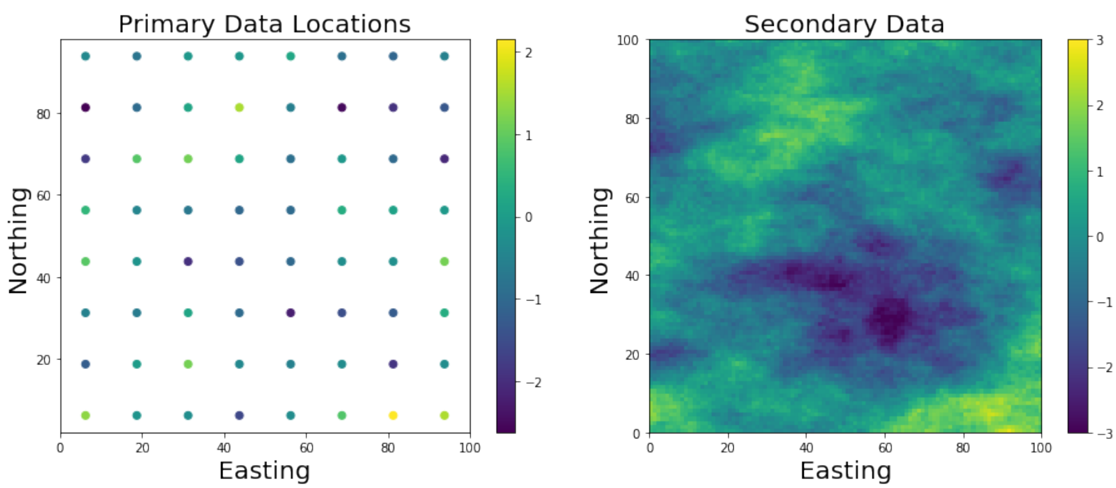
[2]:
```
datafl = pd.read_csv('Data/Cluster1.out')
datafl_sec = pd.read_csv('Data/Ydata.out')
truth = pd.read_csv('Data/true.out')
x = np.asarray(pd.read_csv('Data/X.out')).reshape(len(datafl_sec))
y = np.asarray(pd.read_csv('Data/Y.out')).reshape(len(datafl_sec))
print(datafl.describe())
print(datafl_sec.describe())
```

```
                X          Y    Primary  Secondary
count  64.000000  64.000000  64.000000  64.000000
mean   50.000000  50.000000  -0.424719  -0.197425
std    28.867513  28.867513   1.032475   0.912744
min     6.250000   6.250000  -2.623800  -2.020500
25%    28.125000  28.125000  -1.015100  -0.767850
50%    50.000000  50.000000  -0.387650  -0.224450
75%    71.875000  71.875000   0.176975   0.254700
max    93.750000  93.750000   2.148700   1.889200
                  X             Y     Secondary
count  10000.000000  10000.000000  10000.000000
mean      50.000000     50.000000     -0.250033
std       28.867513     28.867513      0.936804
min        0.500000      0.500000     -3.272810
25%       25.250000     25.250000     -0.817388
50%       50.000000     50.000000     -0.183005
75%       74.750000     74.750000      0.327832
max       99.500000     99.500000      2.781670
```

1.1 Map the Data

[3]:
```
vlim = (-3,3)
f, ax = plt.subplots(1,2,figsize=(13,5.5))
XMIN, XMAX = 0, 100
YMIN, YMAX = 0, 100
SMIN, SMAX = -3,3
gridd = pd.DataFrame()
gridd['Y'] = y
gridd['X'] = x
gridd['Estimate'] = datafl_sec['Secondary']
gridded = np.reshape(gridd.sort_values(by=['Y', 'X'], axis=0,␣
 ↪ascending=True)['Estimate'].values,
                 [100, 100], order='C',)
img0 = ax[0].scatter(datafl['X'],datafl['Y'], c = datafl['Primary'].values)
ax[0].set_title('Primary Data Locations',size = 20)
ax[0].set_xlabel('Easting',size = 20)
ax[0].set_ylabel('Northing',size = 20)
```

```
ax[0].axis('equal')
ax[0].set(xlim=(0, 100), ylim=(0, 100))
img1 = ax[1].imshow(gridded, origin='lower', extent=[XMIN, XMAX, YMIN, YMAX],␣
 ↪aspect='equal',
                interpolation='none', vmin=SMIN, vmax=SMAX, cmap='viridis')
ax[1].set_xlabel('Easting',size = 20)
ax[1].set_ylabel('Northing',size = 20)
ax[1].set_title('Secondary Data',size = 20)
f.colorbar(img0,ax = ax[0])
f.colorbar(img1,ax = ax[1])
plt.tight_layout()
plt.savefig('../0-Figures/Truth',bbox_inches = 'tight',pad_inches = 0)
```



1.2 Check the Distribution and Correlation of the Data

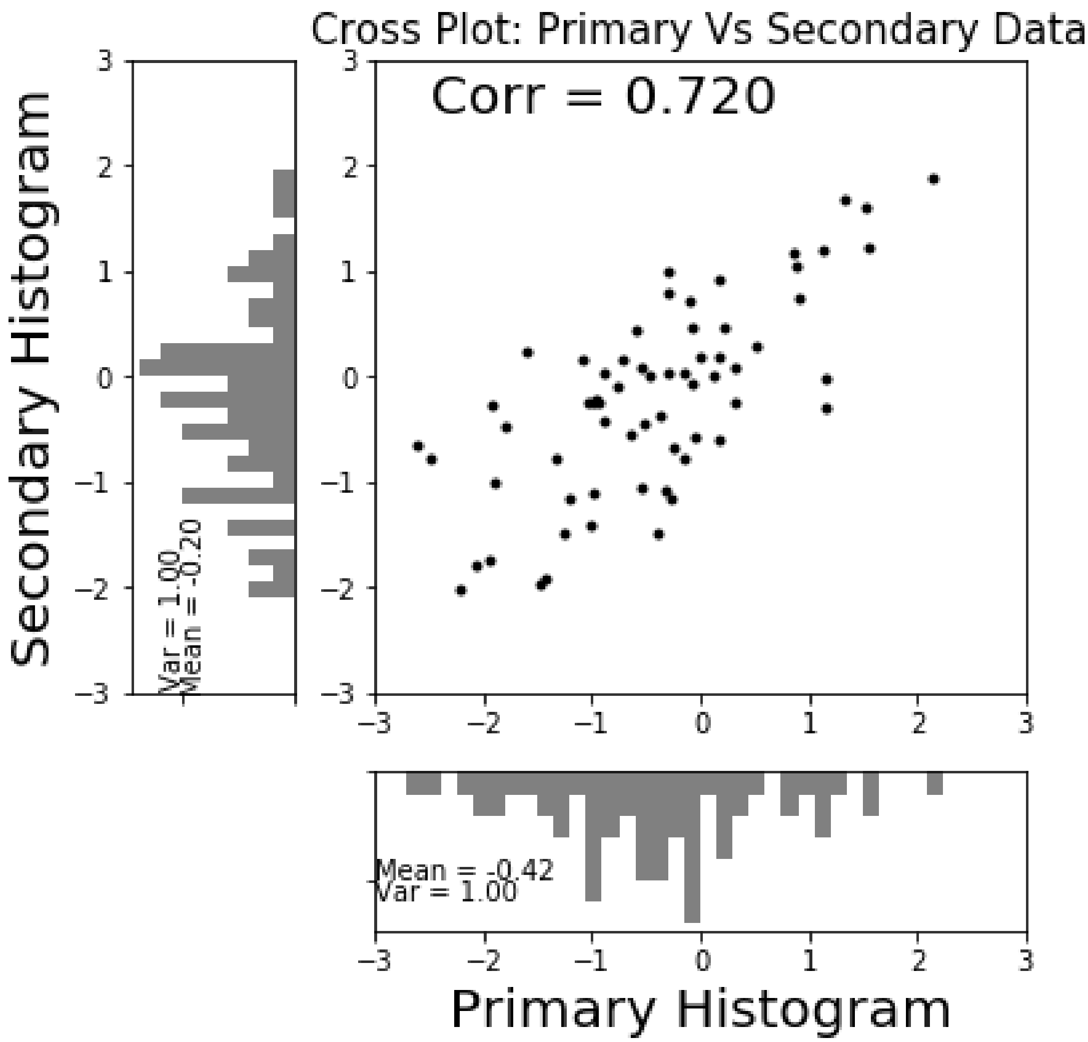
[4]:
```
# Set up the axes with gridspec
corr = np.corrcoef(datafl['Primary'],datafl['Secondary'])[0,1]
vlim= (-3,3)
fig = plt.figure(figsize=(6, 6))
grid = plt.GridSpec(4, 4, hspace=0.5, wspace=0.5)
main_ax = fig.add_subplot(grid[:-1, 1:])
y_hist = fig.add_subplot(grid[:-1, 0], xticklabels=[], sharey=main_ax)
x_hist = fig.add_subplot(grid[-1, 1:], yticklabels=[], sharex=main_ax)

# scatter points on the main axes
main_ax.plot(datafl['Primary'], datafl['Secondary'], 'ok', markersize=3)
main_ax.set_xlim(vlim)
main_ax.set_ylim(vlim)
main_ax.set_title('Cross Plot: Primary Vs Secondary Data',size = 15)
```

```python
main_ax.text(-2.5,2.5,'Corr = {0:.3f}'.format(np.
 ↪corrcoef(datafl['Primary'],datafl['Secondary'])[0,1]),size=20)

# histogram on the attached axes
x_hist.hist(datafl['Primary'], 40, histtype='stepfilled',label = 'Primary',
            orientation='vertical', color='gray',range=vlim)
x_hist.set_xlabel('Primary Histogram',size = 20)
x_hist.invert_yaxis()
x_hist.text(-3,5,'Mean = {0:.2f}'.format(np.average(datafl['Primary'])),size=10)
x_hist.text(-3,6,'Var = {0:.2f}'.format(1.00),size=10)


y_hist.hist(datafl['Secondary'], 40, histtype='stepfilled',
orientation='horizontal', color='gray',range=vlim)
y_hist.set_ylabel('Secondary Histogram',size = 20)
y_hist.invert_xaxis()
y_hist.text(5,-1.5,'Mean = {0:.2f}'.format(np.
 ↪average(datafl['Secondary'])),rotation=90,size=10)
y_hist.text(6,-1.8,'Var = {0:.2f}'.format(1.00),rotation=90,size=10)
plt.savefig('../0-Figures/Cross_plt')
```



2 Correlograms

2.1 Initialize Correlogram Types

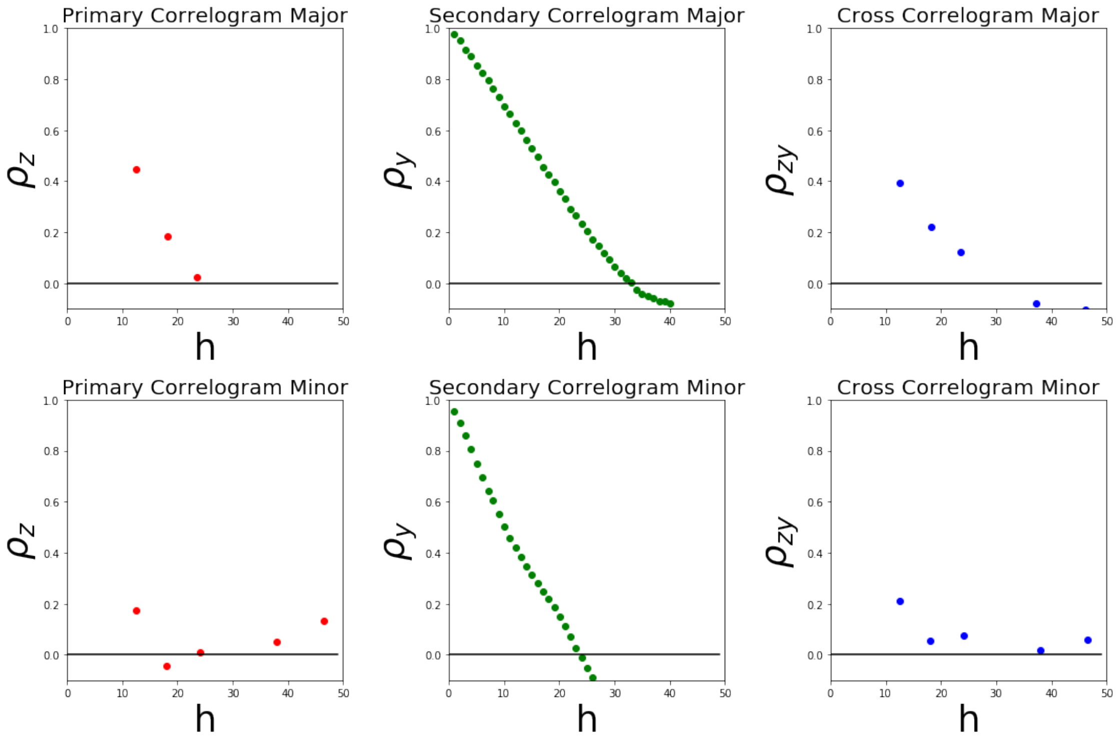
[5]:
```
def covar ( t, d, r ):
    h = d / r
    if t == 1: #Spherical
        c = 1 - h * (1.5 - 0.5 * np.square(h))
        c[h > 1] = 0
    elif t == 2: #Exponential
        c = np.exp( -3 * h )
    elif t == 3: #Gaussian
        c = np.exp( -3 * np.square(h) )
    return c
```

2.2 Fit Experimental Correlogram Points

Experimental variogram points where pre calculated.

[6]:
```
varcalcfl_1 = pd.read_csv('2-vargfls/varcalc_Cluster.out')
varcalcfl_2 = pd.read_csv('2-vargfls/varcalc_YDATA.out')
varcalcfl_3 = pd.read_csv('2-vargfls/varcalc_Cross.out')
```

[7]:
```
ones = np.zeros(shape=(50))
Cross_ones = np.zeros(shape=(50))
H = np.zeros(shape=(50))
Corr_labels = ['Primary Correlogram','Secondary Correlogram','Cross␣
 ↪Correlogram']
Directions = ['Major','Minor']
colors = ['Red','Green','Blue']
labels_2 = ['$\u03C1_{z}$','$\u03C1_{y}$','$\u03C1_{zy}$']
Sill_vals = [1,1,corr]
for h in range(1,50):
    H[h] = h
fig, axes = plt.subplots(2,3, figsize=(15,10))
for i in range(0,3):
    var = locals()['varcalcfl_{}'.format(i+1)]
    for j in range(0,2):
        axes[j,i].plot(var['Lag Distance'][var['Variogram Index']==(j+1)]
                       ,Sill_vals[i]-var['Variogram Value'][var['Variogram␣
 ↪Index']==(j+1)]
                       ,'ro',color =colors[i])
        axes[j,i].set_ylabel(labels_2[i],size=35)
        axes[j,i].plot(H,ones,color = 'Black')
        axes[j,i].set_xlabel('h',size=35)
        axes[j,i].set_title(Corr_labels[i]+' '+Directions[j],size = 20)
plt.setp(axes, xlim=(0,50), ylim=(-0.1,1))
plt.tight_layout()
#plt.savefig('../0-Figures/True_vargs')
```



2.3 Calculate Rotation Matrix

Using a major direction of 90 degrees east of north, this is farily obvious from the primary and secondary data. See http://www.geostatisticslessons.com/lessons/anglespecification

[8]:
```
def Rot_Mat(Azimuth,a_max,a_min):
    theta = (Azimuth/180.) * np.pi
    Rot_Mat = np.dot(np.array([[1/a_max,0],[0,1/a_min]]),
                np.array([[np.cos(theta),np.sin(theta)],[-np.sin(theta),np.
↪cos(theta)]]))
    return Rot_Mat
```

2.4 Primary Correlogram

[9]:
```
# h1 = Set of points X,Y
# h2 = Set of points X,Y
# k = 0 used for calculating the distance between the same points
# k = 1 used for calculationg distance between different points
# k = 2 used for plotting in the major direction
# k = 3 used for plotting in the minor direction
def C_Z(h1,h2,k):
    C = []
    nstruct = 1
```

```
    vtype = [3]
    a_max = [24]
    a_min = [16]
    Azimuth = 90
    cc = [1]
    c= 0
    for i in range(nstruct):
        Q1 = h1.copy()
        Q2 = h2.copy()
        if k == 0:
            d = distance_matrix(np.
↪matmul(Q1,Rot_Mat(Azimuth,a_max[i],a_min[i])),
                                np.
↪matmul(Q2,Rot_Mat(Azimuth,a_max[i],a_min[i])))
        elif k == 1:
            d = np.sqrt(np.square((np.
↪matmul(Q1,Rot_Mat(Azimuth,a_max[i],a_min[i])))-
                              np.tile((np.
↪matmul(Q2,Rot_Mat(Azimuth,a_max[i],a_min[i]))),(k,1))).sum(axis=1))
            d = np.asarray(d).reshape(len(d))
        elif k == 2:
            d = Q1/a_max[i]
        elif k == 3:
            d = Q1/a_min[i]
        c = c + covar(vtype[i],d,1)*cc[i]
    return c
```

2.5 Secondary Correlogram

[10]:
```
# h1 = Set of points X,Y
# h2 = Set of points X,Y
# k = 0 used for calculating the distance between the same points
# k = 1 used for calculationg distance between different points
# k = 2 used for plotting in the major direction
# k = 3 used for plotting in the minor direction
def C_Y(h1,h2,k):
    C = []
    nstruct = 2
    vtype = [1,3]
    a_max = [42,43]
    a_min = [28.5,30]
    Azimuth = 90
    cc = [0.9,0.1]
    c= 0
    for i in range(nstruct):
        Q1 = h1.copy()
```

```
        Q2 = h2.copy()
        if k == 0:
            d = distance_matrix(np.
↪matmul(Q1,Rot_Mat(Azimuth,a_max[i],a_min[i])),
                              np.
↪matmul(Q2,Rot_Mat(Azimuth,a_max[i],a_min[i])))
        elif k == 1:
            d = np.sqrt(np.square((np.
↪matmul(Q1,Rot_Mat(Azimuth,a_max[i],a_min[i])))-
                            np.tile((np.
↪matmul(Q2,Rot_Mat(Azimuth,a_max[i],a_min[i]))),(k,1)))).sum(axis=1)
            d = np.asarray(d).reshape(len(d))
        elif k == 2:
            d = Q1/a_max[i]
        elif k == 3:
            d = Q1/a_min[i]
        c = c + covar(vtype[i],d,1)*cc[i]
    return c
```

2.6 Scaling Correlogram

[11]:
```
# h1 = Set of points X,Y
# h2 = Set of points X,Y
# k = 0 used for calculating the distance between the same points
# k = 1 used for calculationg distance between different points
# k = 2 used for plotting in the major direction
# k = 3 used for plotting in the minor direction
def C_r(h1,h2,k):
    C = []
    nstruct = 1
    vtype = [3]
    a_max = [18]
    a_min = [13]
    Azimuth = 90
    cc = [1]
    c= 0
    for i in range(nstruct):
        Q1 = h1.copy()
        Q2 = h2.copy()
        if k == 0:
            d = distance_matrix(np.
↪matmul(Q1,Rot_Mat(Azimuth,a_max[i],a_min[i])),
                              np.
↪matmul(Q2,Rot_Mat(Azimuth,a_max[i],a_min[i])))
        elif k == 1:
```

```
            d = np.sqrt(np.square((np.
↪matmul(Q1,Rot_Mat(Azimuth,a_max[i],a_min[i])))-
                              np.tile((np.
↪matmul(Q2,Rot_Mat(Azimuth,a_max[i],a_min[i]))),(k,1))).sum(axis=1))
            d = np.asarray(d).reshape(len(d))
        elif k == 2:
            d = Q1/a_max[i]
        elif k == 3:
            d = Q1/a_min[i]
        c = c + covar(vtype[i],d,1)*cc[i]
    return c
```

2.7 C_Z Correlogram MM2

[12]:
```
# h1 = Set of points X,Y
# h2 = Set of points X,Y
# Corr = correlation between primary and secondary data
# k = 0 used for calculating the distance between the same points
# k = 1 used for calculationg distance between different points
# k = 2 used for plotting in the major direction
# k = 3 used for plotting in the minor direction
def C_Z_MM2(h1,h2,k,corr):
    return ((C_Y(h1,h2,k) * corr**2) + ((1-corr**2) * C_r(h1,h2,k)))
```

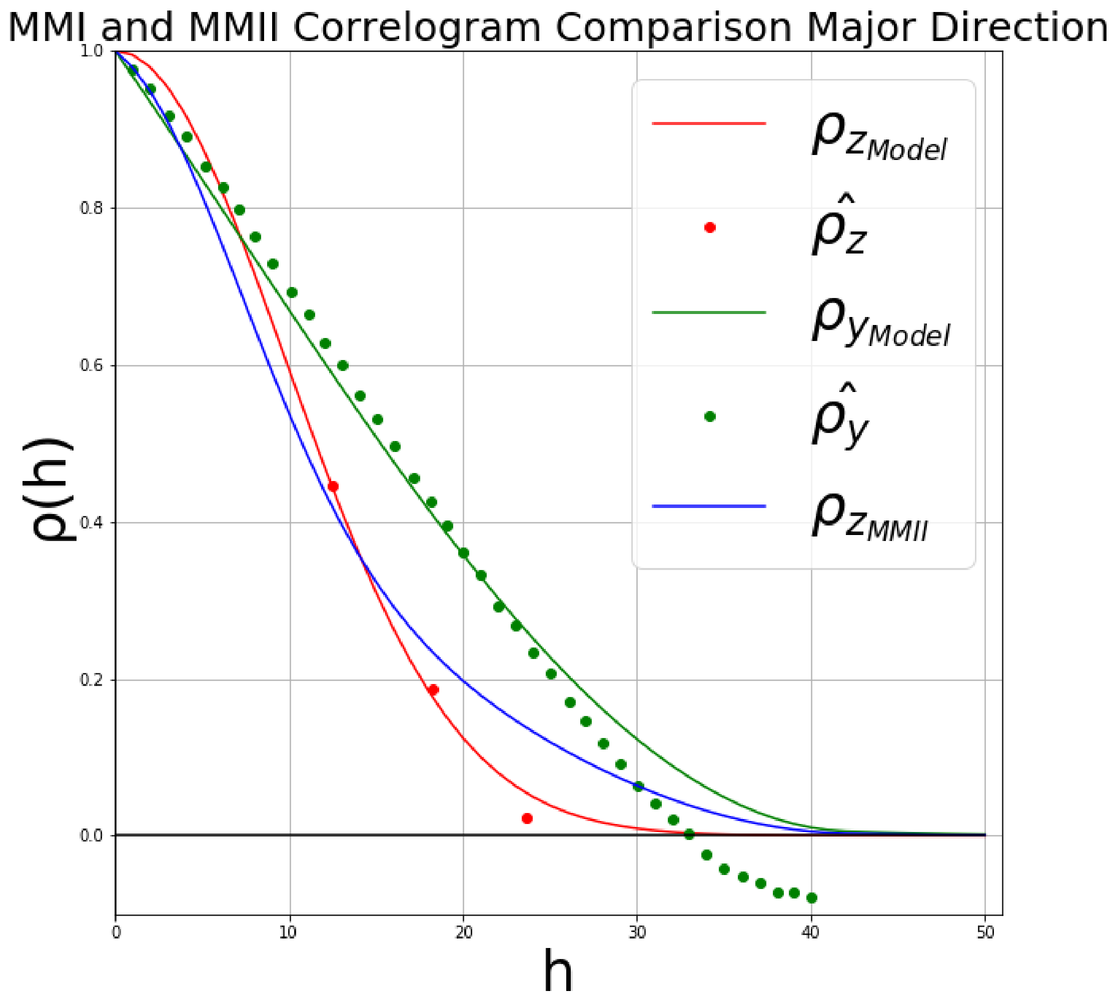
2.8 Plots Correlogram Models

[16]:
```
#Define some matrices for storing variogram values
cy = np.zeros(shape=(51))
cz_True = np.zeros(shape=(51))
cr = np.zeros(shape=(51))
cz = np.zeros(shape=(51))
czy = np.zeros(shape=(51))
H =np.zeros(shape=(51))
ones = np.zeros(shape=(51))
cy_LMC = np.zeros(shape=(51))
cz_LMC = np.zeros(shape=(51))
czy_LMC = np.zeros(shape=(51))
#Define some plotting labels
labels_1 = ['$\hat{\u03C1_{z}}$','$\hat{\u03C1_{y}}$','$\hat{\u03C1_{zy}}$']
labels_MM =␣
↪['$\u03C1_{z_{Model}}$','$\u03C1_{y_{Model}}$','$\u03C1_{z_{MMII}}$']
labels_lmc = ['$\u03C1_{z_{LMC}}$','$\u03C1_{y_{LMC}}$','$\u03C1_{zy_{LMC}}$']
colors_lmc = ['Orange','Yellow','Grey']
```

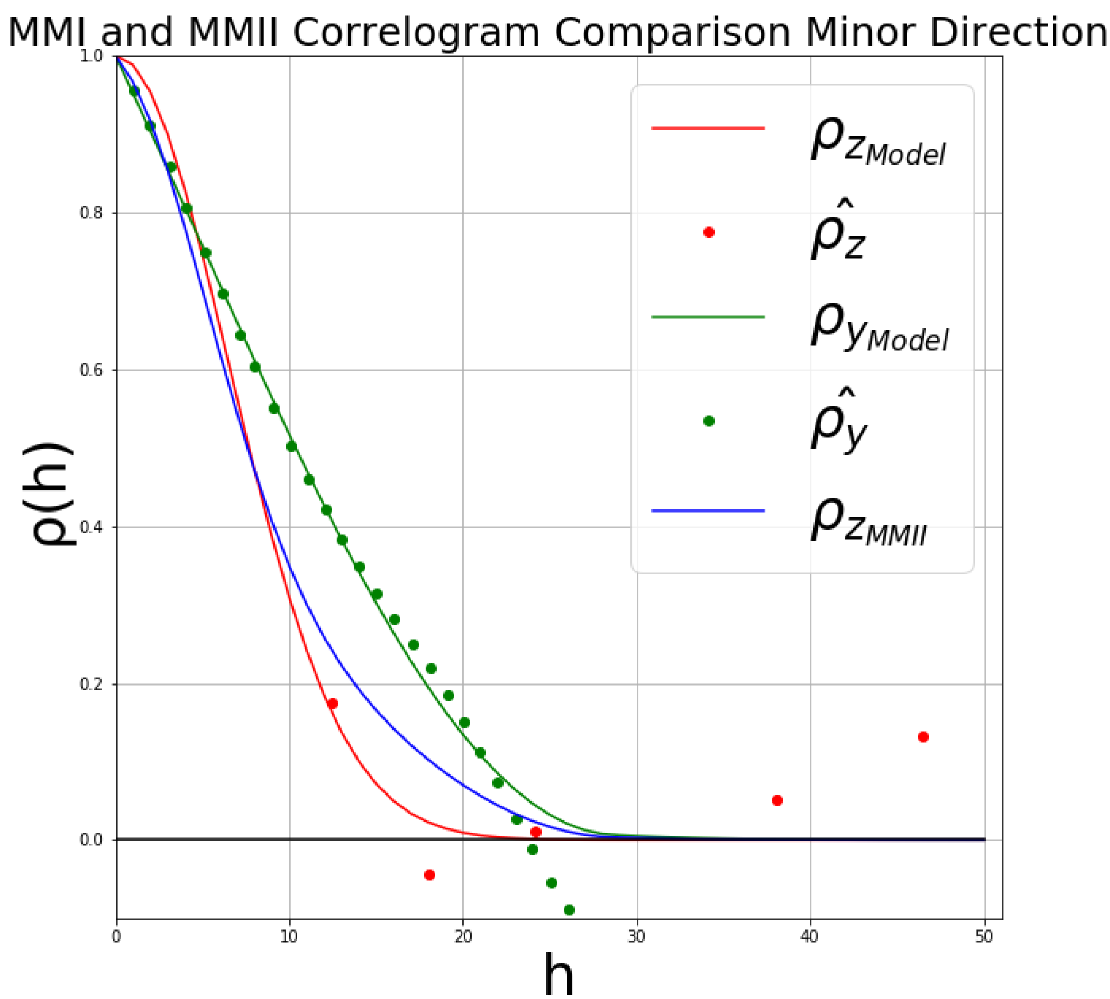
[17]:
```
varg_type = 2 #See Correlogram Functions
for Dir in Directions:
```

```
    for h in range(0,51):
        cy[h] = C_Y(np.matrix(h),np.matrix(h),varg_type)
        cz_True[h] = C_Z(np.matrix(h),np.matrix(h),varg_type)
        cz[h] = C_Z_MM2(np.matrix(h),np.matrix(h),varg_type,corr)
        cr[h] = C_r(np.matrix(h),np.matrix(h),varg_type)
        H[h] = h
    MM_vargs = [cz_True,cy,cz]
    fig, axes = plt.subplots(1,1, figsize=(10,10))
    for i in range(0,3):
        axes.plot(H,MM_vargs[i],color = colors[i],label = labels_MM[i])
        if((i+1)<3):
            var = locals()['varcalcfl_{}'.format(i+1)]
            axes.plot(var['Lag Distance'][var['Variogram Index']==(varg_type-1)]
                               ,Sill_vals[i]-var['Variogram Value'][var['Variogram␣
↪Index']==(varg_type-1)]
                               ,'ro',color =colors[i],label = labels_1[i])


    axes.plot(H,ones,color = 'Black')
    axes.grid()
    plt.xlim(0,51)
    plt.ylim(-0.1,1)
    plt.ylabel('\u03C1(h)',size=35)
    plt.xlabel('h',size=35)
    plt.title('MMI and MMII Correlogram Comparison {} Direction'.
↪format(Dir),size = 25)
    axes.legend(loc='best', prop={"size":35})
    plt.savefig('../0-Figures/MM1_MM2_var_{}'.format(Dir))
    varg_type = varg_type+1
```



MMI and MMII Correlogram Comparison Major Direction
ρ(h)
h
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
$\rho_{z_{Model}}$
$\hat{\rho_z}$
$\rho_{y_{Model}}$
$\hat{\rho_y}$
$\rho_{z_{MMII}}$



2.9 Cross Correlogram

[15]:
```
def C_ZY(h1,h2,k,corr):
    C = []
    nstruct = 1
    vtype = [1]
    a_max = [45]
    a_min = [30]
    Azimuth = 90
    cc = [corr]
    c= 0
    for i in range(nstruct):
        Q1 = h1.copy()
        Q2 = h2.copy()
        if k == 0:
```

```
            d = distance_matrix(np.
↪matmul(Q1,Rot_Mat(Azimuth,a_max[i],a_min[i])),
                                    np.
↪matmul(Q2,Rot_Mat(Azimuth,a_max[i],a_min[i])))
        elif k == 1:
            d = np.sqrt(np.square((np.
↪matmul(Q1,Rot_Mat(Azimuth,a_max[i],a_min[i])))-
                                  np.tile((np.
↪matmul(Q2,Rot_Mat(Azimuth,a_max[i],a_min[i]))),(k,1))).sum(axis=1))
            d = np.asarray(d).reshape(len(d))
        elif k == 2:
            d = Q1/a_max[i]
        elif k == 3:
            d = Q1/a_min[i]
        c = c + covar(vtype[i],d,1)*cc[i]
    return c
```

2.10 LMC

The new variogram that will be used for full cokriging, these variograms will be slightly differnt the variograms modelled above. For LMC variograms the sill should be the variance of the variable for the primary and secondary variables. The correlations is the sill of the cross-correlogram

2.10.1 Primary

[16]:

```
# h1 = Set of points X,Y
# h2 = Set of points X,Y
# k = 0 used for calculating the distance between the same points
# k = 1 used for calculationg distance between different points
# k = 2 used for plotting in the major direction
# k = 3 used for plotting in the minor direction
def C_Z_LMC(h1,h2,k):
    C = []
    nstruct = 2
    vtype = [1,1]
    a_max = [33,40]
    a_min = [15,30]
    Azimuth = 90
    cc = [0.85,0.15]
    c = 0
    for i in range(nstruct):
        Q1 = h1.copy()
        Q2 = h2.copy()
        if k == 0:
            d = distance_matrix(np.
↪matmul(Q1,Rot_Mat(Azimuth,a_max[i],a_min[i])),
```

```
                                        np.
↪matmul(Q2,Rot_Mat(Azimuth,a_max[i],a_min[i])))
        elif k == 1:
            d = np.sqrt(np.square((np.
↪matmul(Q1,Rot_Mat(Azimuth,a_max[i],a_min[i])))-
                              np.tile((np.
↪matmul(Q2,Rot_Mat(Azimuth,a_max[i],a_min[i]))),(k,1))).sum(axis=1))
            d = np.asarray(d).reshape(len(d))
        elif k == 2:
            d = Q1/a_max[i]
        elif k == 3:
            d = Q1/a_min[i]
        c = c + covar(vtype[i],d,1)*cc[i]
    return c
```

2.10.2 Secondary

[17]:
```
# h1 = Set of points X,Y
# h2 = Set of points X,Y
# k = 0 used for calculating the distance between the same points
# k = 1 used for calculationg distance between different points
# k = 2 used for plotting in the major direction
# k = 3 used for plotting in the minor direction
def C_Y_LMC(h1,h2,k):
    C = []
    nstruct = 2
    vtype = [1,1]
    a_max = [33,40]
    a_min = [15,30]
    Azimuth = 90
    cc = [0.25,0.75]
    c= 0
    for i in range(nstruct):
        Q1 = h1.copy()
        Q2 = h2.copy()
        if k == 0:
            d = distance_matrix(np.
↪matmul(Q1,Rot_Mat(Azimuth,a_max[i],a_min[i])),
                                        np.
↪matmul(Q2,Rot_Mat(Azimuth,a_max[i],a_min[i])))
        elif k == 1:
            d = np.sqrt(np.square((np.
↪matmul(Q1,Rot_Mat(Azimuth,a_max[i],a_min[i])))-
                              np.tile((np.
↪matmul(Q2,Rot_Mat(Azimuth,a_max[i],a_min[i]))),(k,1))).sum(axis=1))
            d = np.asarray(d).reshape(len(d))
```

```
        elif k == 2:
            d = Q1/a_max[i]
        elif k == 3:
            d = Q1/a_min[i]
        c = c + covar(vtype[i],d,1)*cc[i]
    return c
```

2.10.3 Cross

[18]:
```
# h1 = Set of points X,Y
# h2 = Set of points X,Y
# Corr = correlation between primary and secondary data
# k = 0 used for calculating the distance between the same points
# k = 1 used for calculationg distance between different points
# k = 2 used for plotting in the major direction
# k = 3 used for plotting in the minor direction
def C_ZY_LMC(h1,h2,k,corr):
    C = []
    nstruct = 2
    vtype = [1,1]
    a_max = [33,40]
    a_min = [15,30]
    Azimuth = 90
    cc = [corr*0.6,corr*0.4]
    c= 0
    for i in range(nstruct):
        Q1 = h1.copy()
        Q2 = h2.copy()
        if k == 0:
            d = distance_matrix(np.
 ↪matmul(Q1,Rot_Mat(Azimuth,a_max[i],a_min[i])),
                                np.
 ↪matmul(Q2,Rot_Mat(Azimuth,a_max[i],a_min[i])))
        elif k == 1:
            d = np.sqrt(np.square((np.
 ↪matmul(Q1,Rot_Mat(Azimuth,a_max[i],a_min[i])))-
                              np.tile((np.
 ↪matmul(Q2,Rot_Mat(Azimuth,a_max[i],a_min[i]))),(k,1))).sum(axis=1))
            d = np.asarray(d).reshape(len(d))
        elif k == 2:
            d = Q1/a_max[i]
        elif k == 3:
            d = Q1/a_min[i]
        c = c + covar(vtype[i],d,1)*cc[i]
    return c
```

[19]:
```
s_1 = np.array([[0.0001, 0],
                [0, 0.0001]])
s_2 = np.array([[0.85, corr*0.6],
                [corr*0.6,0.25]])
s_3 = np.array([[0.15, corr*0.4],
                [corr*0.4,0.75]])

print(np.linalg.det(s_1))
print(np.linalg.det(s_2))
print(np.linalg.det(s_3))
print(s_1+s_2+s_3)
```

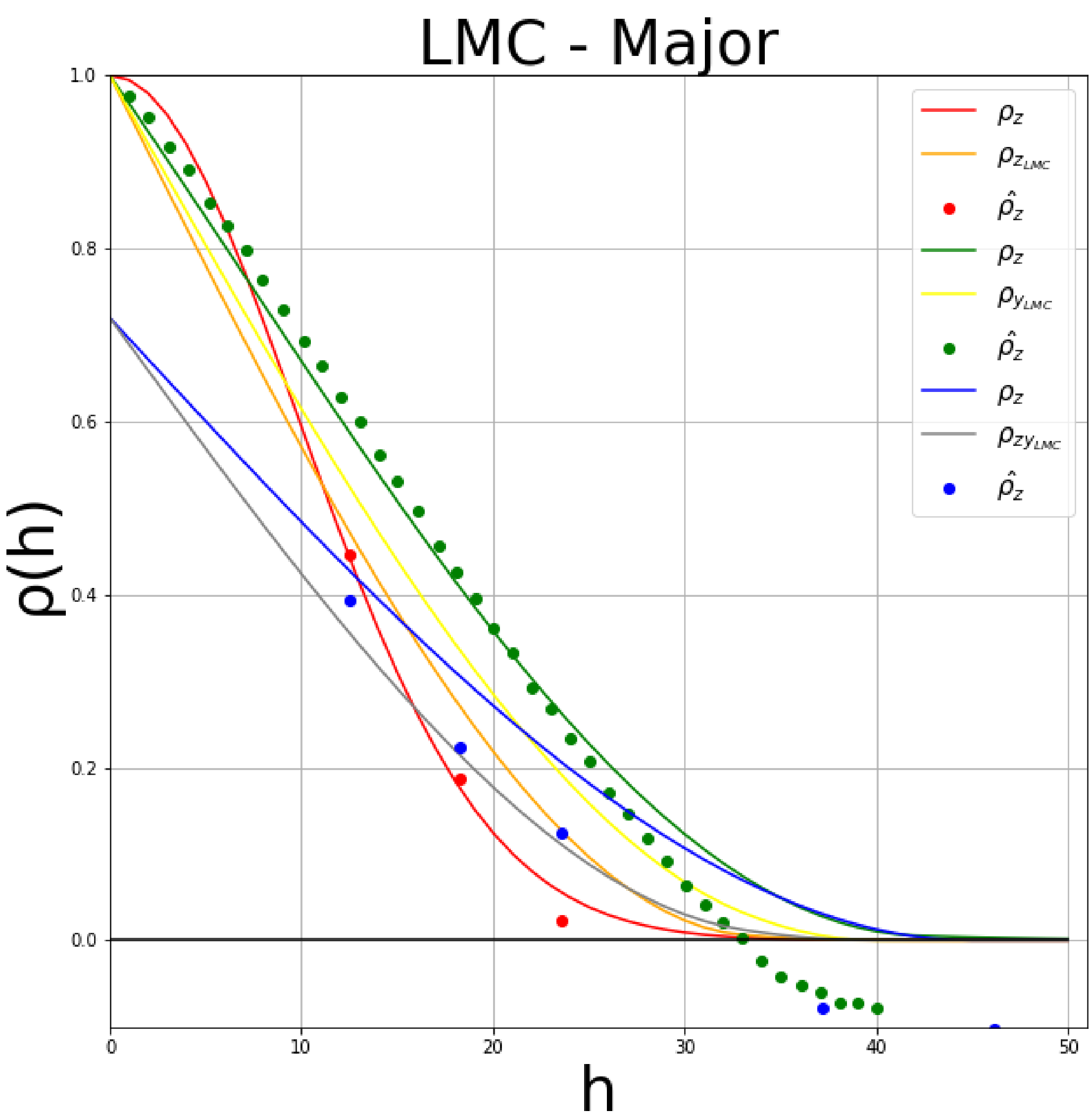
```
1.0000000000000018e-08
0.02601118558622158
0.02961608248276517
[[1.   0.72]
 [0.72 1.  ]]
```

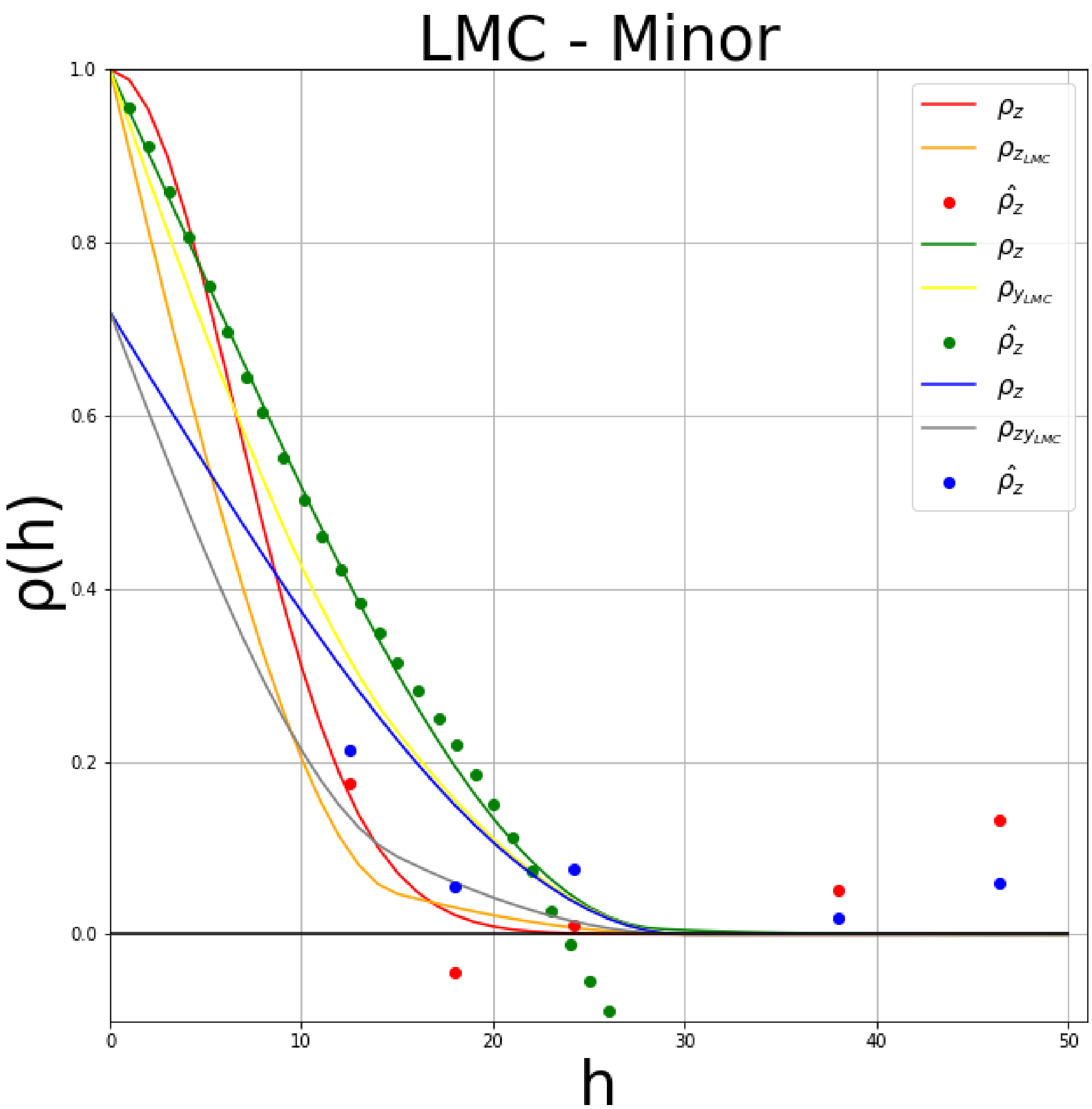
2.10.4 Plot LMC

[20]:
```
varg_type = 2 #See Correlogram Functions
for Dir in Directions:
    for h in range(0,51):
        cy[h] = C_Y(np.matrix(h),np.matrix(h),varg_type)
        cz_True[h] = C_Z(np.matrix(h),np.matrix(h),varg_type)
        czy[h] = C_ZY(np.matrix(h),np.matrix(h),varg_type,corr)
        cy_LMC[h] = C_Y_LMC(np.matrix(h),np.matrix(h),varg_type)
        cz_LMC[h] = C_Z_LMC(np.matrix(h),np.matrix(h),varg_type)
        czy_LMC[h] = C_ZY_LMC(np.matrix(h),np.matrix(h),varg_type,corr)
    Vargs = [cz_True,cy,czy]
    LMCS = [cz_LMC,cy_LMC,czy_LMC]
    fig, axes = plt.subplots(1,1, figsize=(10,10))
    for i in range(0,3):
        var = locals()['varcalcfl_{}'.format(i+1)]
        axes.plot(H,Vargs[i],color= colors[i],label = labels_2[0])
        axes.plot(H,LMCS[i],color=colors_lmc[i],label = labels_lmc[i])
        axes.plot(var['Lag Distance'][var['Variogram Index']==(varg_type-1)]
                   ,Sill_vals[i]-var['Variogram Value'][var['Variogram␣
↪Index']==(varg_type-1)]
                   ,'ro',color =colors[i],label = labels_1[0])

    axes.plot(H,ones,color = 'Black')
    axes.grid()
    plt.xlim(0,51)
    plt.ylim(-0.1,1)
    plt.ylabel('\u03C1(h)',size=35)
    plt.xlabel('h',size=35)
```

```
    plt.title('LMC - {}'.format(Dir),size = 35)
    axes.legend(loc='best', prop={"size":15})
    #plt.savefig('../0-Figures/MM1_MM2_var_{}'.format(Dir))
    varg_type = varg_type+1
```





3 Kriging

3.1 Data Statistics

[21]:
```
Mean_Z = np.average(datafl['Primary'])
STD_Z = 1.0
print(Mean_Z)
print(STD_Z)
```

```
-0.42471875000000003
1.0
```

[22]:
```
Mean_Y = np.average(datafl['Secondary'])
STD_Y = 1.0
print(Mean_Y)
print(STD_Y)
```

```
-0.19742499999999996
1.0
```

[23]:
```
corr = np.corrcoef(datafl['Primary'],datafl['Secondary'])[0,1]
print(corr)
```

```
0.7197391780935075
```

3.2 Create a KDTree to Quickly Get Nearest Points

[24]:
```
from sklearn.neighbors import KDTree
```

[25]:
```
datafl_XY = datafl.as_matrix(['X', 'Y'])
tree = KDTree(datafl_XY)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1:
FutureWarning: Method .as_matrix will be removed in a future version. Use
.values instead.
  """Entry point for launching an IPython kernel.
```

[26]:
```
Pred_grid_xy = np.matrix([x,y]).T
```

[27]:
```
#Primary Data Search for Kriging
k = 60 #number of data to use
X_Y = np.zeros((len(x),k,2))
X_Y_Star = np.zeros((len(x),k,2))
closematrix_Primary = np.zeros((len(x),k))
closematrix_Secondary = np.zeros((len(x),k))
neardistmatrix = np.zeros((len(x),k))
for i in range (0,len(x)):
    nearest_dist, nearest_ind = tree.query(Pred_grid_xy[i:i+1,:], k=k)
    a  = nearest_ind.ravel()
    group = datafl.iloc[a,:]
    closematrix_Primary[i,:] = group['Primary']
    closematrix_Secondary[i,:] = group['Secondary']
    neardistmatrix[i,:] = nearest_dist
    X_Y[i,:,:] = group[['X','Y']]
```

[28]:
```
datafl_XY_2nd = datafl_sec.as_matrix(['X', 'Y'])
tree_2nd = KDTree(datafl_XY_2nd)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1:
FutureWarning: Method .as_matrix will be removed in a future version. Use
```

```
.values instead.
  """Entry point for launching an IPython kernel.
```

[29]:
```
#Secondary Data Search for CoKriging
k = k #number of neighbours
X_Y_2nd = np.zeros((len(x),k,2))
closematrix_Secondary_2nd = np.zeros((len(x),k))
for i in range (0,len(x)):
    nearest_dist, nearest_ind = tree_2nd.query(Pred_grid_xy[i:i+1,:], k=k)
    a  = nearest_ind.ravel()
    group = datafl_sec.iloc[a,:]
    closematrix_Secondary_2nd[i,:] = group['Secondary']
    X_Y_2nd[i,:,:] = group[['X','Y']]
```

3.3 Simple Kriging

[30]:
```
est_SK = np.zeros(shape = (len(x)))
for z in tqdm(range(0,len(x))):
    Kriging_Matrix = np.zeros(shape=((k,k)))
    #h = distance_matrix(X_Y[z,:,:].tolist(),X_Y[z,:,:].tolist())
    #C_ZZ
    Kriging_Matrix = C_Z(X_Y[z,:,:],X_Y[z,:,:],0)
    #Set up Right Hand Side
    #print(Kriging_Matrix.reshape(((k)),((k))))
    r = np.zeros(shape=(k))
    k_weights = r
    #RHS #C_z*
    r = C_Z(X_Y[z,:,:],np.tile(Pred_grid_xy[z],(k,1)),1)
    Kriging_Matrix.reshape(((k)),((k)))
    #Calculate Kriging Weights
    try:
        k_weights = np.dot(np.linalg.inv(Kriging_Matrix),r)
    except:
        s_m = s_m+1
        sm_idx.append(z)
        k_weights = np.dot(scipy.linalg.pinv(Kriging_Matrix),r)
    #Start Est at zero
    est_SK[z] = 0
    #add in mean_z
    est_SK[z] = est_SK[z] + Mean_Z
    for i in range (0,k):
        #add in Z_i
        est_SK[z] = est_SK[z] + k_weights[i]*(closematrix_Primary[z,i] - Mean_Z)
#print(Kriging_Matrix.reshape(((2*k)+1),((2*k)+1)))
```

```
100%|
| 10000/10000 [00:33<00:00, 302.63it/s]
```

3.4 Full Cokriging

[31]:
```
cz = np.zeros(shape = (k,k))
czy = np.zeros(shape = (k,k))
czy_2 = np.zeros(shape = (k,k))
cy = np.zeros(shape = (k,k))
s_m = 0
sm_idx = []
est_Full_CCK = np.zeros(shape = (len(x)))
for z in tqdm(range(0,len(x))):
    Kriging_Matrix = np.zeros(shape=((k*2),(k*2)))
    #C_ZZ
    cz = C_Z_LMC(X_Y[z,:,:],X_Y[z,:,:],0)
    #C_ZY
    czy = C_ZY_LMC(X_Y[z,:,:],X_Y_2nd[z,:,:],0,corr)
    czy_2 = C_ZY_LMC(X_Y_2nd[z,:,:],X_Y[z,:,:],0,corr)
    #C_YY
    cy = C_Y_LMC(X_Y_2nd[z,:,:],X_Y_2nd[z,:,:],0)
    Kriging_Matrix = np.vstack((np.hstack((cz,czy)),np.hstack((czy.T,cy))))
    #print(Kriging_Matrix)
    #Set up Right Hand Sides
    r = np.zeros(shape=(k*2))
    k_weights = np.zeros(shape=(k*2))
    #RHS #C_z*
    r[0:k] = C_Z_LMC(X_Y[z,:,:],np.tile(Pred_grid_xy[z],(k,1)),1)
    #RHS #C_zy*
    r[k:k*2] = C_ZY_LMC(X_Y_2nd[z,:,:],np.tile(Pred_grid_xy[z],(k,1)),1,corr)
    #Calculate Kriging Weights
    try:
        k_weights = np.dot(np.linalg.inv(Kriging_Matrix),r)
    except:
        s_m = s_m+1
        sm_idx.append(z)
        k_weights = np.dot(scipy.linalg.pinv(Kriging_Matrix),r)
    #Start Est at zero
    est_Full_CCK[z] = 0
    #add in mean_z
    est_Full_CCK[z] = est_Full_CCK[z] + Mean_Z
    for i in range (0,k):
        #add in Z_i
        est_Full_CCK[z] = est_Full_CCK[z] + k_weights[i] *␣
 ↪(closematrix_Primary[z,i] - Mean_Z)/STD_Z
        #add in Y_i
        est_Full_CCK[z] = est_Full_CCK[z] + k_weights[i+k] *␣
 ↪(closematrix_Secondary_2nd[z,i]- Mean_Y)/STD_Y
print('There where {} Singular Matrices'.format(s_m))
#print(Kriging_Matrix.reshape(((2*k)),((2*k))))
```

```
100%|
| 10000/10000 [01:46<00:00, 93.47it/s]

There where 0 Singular Matrices
```

3.5 Simple Collocated Cokriging - MM1

[32]:
```
est_MM1 = np.zeros(shape = (len(x)))
for z in tqdm(range(0,len(x))):
    Kriging_Matrix = np.zeros(shape=((k+1),(k+1)))
    #C_ZZ
    Kriging_Matrix[0:k,0:k] = C_Z(X_Y[z,:,:],X_Y[z,:,:],0)
    #Set up Right Hand Side
    #print(Kriging_Matrix.reshape(((2*k)+1),((2*k)+1)))
    r = np.zeros(shape=(k+1))
    k_weights = np.zeros(shape=(k))
    #RHS #C_z*
    r[0:k] = C_Z(X_Y[z,:,:],np.tile(Pred_grid_xy[z],(k,1)),1)
    #RHS corr
    r[k] = corr
    #c_zy
    Kriging_Matrix[k,0:k+1] = r * corr
    Kriging_Matrix[0:k+1,k] = r * corr
    Kriging_Matrix[k,k] = 1
    #Calculate Kriging Weights
    try:
        k_weights = np.dot(np.linalg.inv(Kriging_Matrix),r)
    except:
        s_m = s_m+1
        sm_idx.append(z)
        k_weights = np.dot(scipy.linalg.pinv(Kriging_Matrix),r)
    #Start Est at zero
    est_MM1[z] = 0
    #add in mean_z
    est_MM1[z] = est_MM1[z] + Mean_Z
    #add in the Y_0
    est_MM1[z] = est_MM1[z] + k_weights[k] *␣
↪(datafl_sec['Secondary'][z]-Mean_Y)/STD_Y
    for i in range (0,k):
        #add in Z_i
        est_MM1[z] = est_MM1[z] + k_weights[i] * (closematrix_Primary[z,i] -␣
↪Mean_Z)/STD_Z
#print(Kriging_Matrix.reshape(((2*k)+1),((2*k)+1)))
```

```
100%|
| 10000/10000 [00:34<00:00, 288.22it/s]
```

3.6 Simple Collocated Cokriging - MM2

[33]:

```
est_MM2 = np.zeros(shape = (len(x)))
s_m = 0
sm_idx =[]
for z in tqdm(range(0,len(x))):
    Kriging_Matrix = np.zeros(shape=((k+1),(k+1)))
    #C_ZZ
    Kriging_Matrix[0:k,0:k] = C_Z_MM2(X_Y[z,:,:],X_Y[z,:,:],0,corr)
    #Set up Right Hand Side
    #print(Kriging_Matrix.reshape(((2*k)+1),((2*k)+1)))
    r = np.zeros(shape=(k+1))
    k_weights = np.zeros(shape=(k))
    #RHS #C_z*
    r[0:k] = C_Z_MM2(X_Y[z,:,:],np.tile(Pred_grid_xy[z],(k,1)),1,corr)
    #RHS corr
    r[k] = corr
    #c_zy
    Kriging_Matrix[k,0:k] = C_Y(X_Y[z,:,:],np.tile(Pred_grid_xy[z],(k,1)),1) *␣
 ↪corr
    Kriging_Matrix[0:k,k] = C_Y(X_Y[z,:,:],np.tile(Pred_grid_xy[z],(k,1)),1) *␣
 ↪corr
    Kriging_Matrix[k,k] = 1
    #Calculate Kriging Weights
    try:
        k_weights = np.dot(np.linalg.inv(Kriging_Matrix),r)
    except:
        s_m = s_m+1
        sm_idx.append(z)
        k_weights = np.dot(scipy.linalg.pinv(Kriging_Matrix),r)
    #Start Est at zero
    est_MM2[z] = 0
    #add in mean_z
    est_MM2[z] = est_MM2[z] + Mean_Z
    #add in the Y_0
    est_MM2[z] = est_MM2[z] + k_weights[k] *␣
 ↪(datafl_sec['Secondary'][z]-Mean_Y)/STD_Y
    for i in range (0,k):
        #add in Z_i
        est_MM2[z] = est_MM2[z] + k_weights[i] * (closematrix_Primary[z,i] -␣
 ↪Mean_Z)/STD_Z
#print(Kriging_Matrix.reshape(((2*k)+1),((2*k)+1)))
print('There where {} Singular Matrices'.format(s_m))
```

```
100%|
| 10000/10000 [00:49<00:00, 202.52it/s]

There where 0 Singular Matrices
```

3.7 Intrinsic Collocated Cokriging - MM1

[34]:
```
s_m = 0
sm_idx =[]
cz = np.zeros(shape = (k,k))
czy = np.zeros(shape = (k,k))
cy = np.zeros(shape = (k,k))
est_icck_MM1 = np.zeros(shape = (len(x)))
for z in tqdm(range(0,len(x))):
    Kriging_Matrix = np.zeros(shape=((k*2+1),(k*2+1)))
    #C_ZZ
    cz = C_Z(X_Y[z,:,:],X_Y[z,:,:],0)
    #C_ZY
    czy = C_Z(X_Y[z,:,:],X_Y[z,:,:],0) * corr
    #C_YY
    cy = C_Z(X_Y[z,:,:],X_Y[z,:,:],0)
    #Set up Right Hand Side
    Kriging_Matrix[0:k*2,0:k*2] = np.vstack((np.hstack((cz,czy)),np.hstack((czy.
 ↪T,cy))))
    #print(Kriging_Matrix.reshape(((2*k)+1),((2*k)+1)))
    r = np.zeros(shape=(k*2)+1)
    k_weights = r
    #RHS #C_z*
    r[0:k] = C_Z(X_Y[z,:,:],np.tile(Pred_grid_xy[z],(k,1)),1)
    #RHS #C_yz*
    r[k:k*2] = C_Z(X_Y[z,:,:],np.tile(Pred_grid_xy[z],(k,1)),1) * corr
    #RHS corr
    r[k*2] = corr
    #c_zy
    Kriging_Matrix[k*2,0:k] = C_Z(X_Y[z,:,:],np.tile(Pred_grid_xy[z],(k,1)),1)␣
 ↪* corr
    Kriging_Matrix[0:k,k*2] = C_Z(X_Y[z,:,:],np.tile(Pred_grid_xy[z],(k,1)),1)␣
 ↪* corr
    #c_z
    Kriging_Matrix[k*2,k:k*2] = C_Z(X_Y[z,:,:],np.tile(Pred_grid_xy[z],(k,1)),1)
    Kriging_Matrix[k:k*2,k*2] = C_Z(X_Y[z,:,:],np.tile(Pred_grid_xy[z],(k,1)),1)
    Kriging_Matrix[k*2,k*2] = 1
    #Calculate Kriging Weights
    try:
        k_weights = np.dot(np.linalg.inv(Kriging_Matrix),r)
    except:
        s_m = s_m+1
        sm_idx.append(z)
        k_weights = np.dot(scipy.linalg.pinv(Kriging_Matrix),r)
    #Start Est at zero
```

```
    est_icck_MM1[z] = 0
    #add in mean_z
    est_icck_MM1[z] = est_icck_MM1[z] + Mean_Z
    #add in the Y_0
    est_icck_MM1[z] = est_icck_MM1[z] + k_weights[k*2] *␣
↪(datafl_sec['Secondary'][z]-Mean_Y)/STD_Y
    for i in range (0,k):
        #add in Z_i
        est_icck_MM1[z] = est_icck_MM1[z] + k_weights[i] *␣
↪(closematrix_Primary[z,i] - Mean_Z)/STD_Z
        #add in Y_i
        est_icck_MM1[z] = est_icck_MM1[z] + k_weights[i+k] *␣
↪(closematrix_Secondary[z,i]- Mean_Y)/STD_Y
#print(Kriging_Matrix.reshape(((2*k)+1),((2*k)+1)))
print('There where {} Singular Matrices'.format(s_m))
```

```
100%|
| 10000/10000 [01:33<00:00, 106.64it/s]

There where 0 Singular Matrices
```

3.8 Intrinsic Collocated Cokriging - MM2

[35]:
```
s_m = 0
sm_idx =[]
cz = np.zeros(shape = (k,k))
czy = np.zeros(shape = (k,k))
cy = np.zeros(shape = (k,k))
est_icck_MM2 = np.zeros(shape = (len(x)))
for z in tqdm(range(0,len(x))):
    Kriging_Matrix = np.zeros(shape=((k*2+1),(k*2+1)))
    #C_ZZ
    #1
    cz = C_Z_MM2(X_Y[z,:,:],X_Y[z,:,:],0,corr)
    #C_ZY
    #2,#3
    czy = corr * C_Y(X_Y[z,:,:],X_Y[z,:,:],0)
    #C_YY
    #4
    cy = C_Y(X_Y[z,:,:],X_Y[z,:,:],0)
    #Set up Right Hand Side
    #print(Kriging_Matrix.reshape(((2*k)+1),((2*k)+1)))
    Kriging_Matrix[0:k*2,0:k*2] = np.vstack((np.hstack((cz,czy)),np.hstack((czy.
↪T,cy))))
    r = np.zeros(shape=(k*2)+1)
    k_weights = r
```

```
    #RHS #C_z*
    #5
    r[0:k] = C_Z_MM2(X_Y[z,:,:],np.tile(Pred_grid_xy[z],(k,1)),1,corr)
    #RHS #C_yz*
    #6
    r[k:k*2] = C_Y(X_Y[z,:,:],np.tile(Pred_grid_xy[z],(k,1)),1) * corr
    #RHS corr
    #7
    r[k*2] = corr
    #c_zy
    #8
    Kriging_Matrix[k*2,0:k] = C_Y(X_Y[z,:,:],np.tile(Pred_grid_xy[z],(k,1)),1)␣
↪* corr
    Kriging_Matrix[0:k,k*2] = C_Y(X_Y[z,:,:],np.tile(Pred_grid_xy[z],(k,1)),1)␣
↪* corr
    #c_y
    #9
    Kriging_Matrix[k*2,k:k*2] = C_Y(X_Y[z,:,:],np.tile(Pred_grid_xy[z],(k,1)),1)
    Kriging_Matrix[k:k*2,k*2]  = C_Y(X_Y[z,:,:],np.
↪tile(Pred_grid_xy[z],(k,1)),1)
    Kriging_Matrix[k*2,k*2] = 1
    #Kriging_Matrix.reshape(((2*k)+1),((2*k)+1))
    #Calculate Kriging Weights
    try:
        k_weights = np.dot(np.linalg.inv(Kriging_Matrix),r)
    except:
        s_m = s_m+1
        sm_idx.append(z)
        k_weights = np.dot(scipy.linalg.pinv(Kriging_Matrix),r)
    #Start Est at zero
    est_icck_MM2[z] = 0
    #add in mean_z
    est_icck_MM2[z] = est_icck_MM2[z] + Mean_Z
    #add in the Y_0
    est_icck_MM2[z] = est_icck_MM2[z] + k_weights[k*2] *␣
↪(datafl_sec['Secondary'][z]-Mean_Y)/STD_Y
    for i in range (0,k):
        #add in Z_i
        est_icck_MM2[z] = est_icck_MM2[z] + k_weights[i] *␣
↪(closematrix_Primary[z,i] - Mean_Z)/STD_Z
        #add in Y_i
        est_icck_MM2[z] = est_icck_MM2[z] + k_weights[i+k] *␣
↪(closematrix_Secondary[z,i]- Mean_Y)/STD_Y
#print(Kriging_Matrix.reshape(((2*k)+1),((2*k)+1)))
print('There where {} Singular Matrices'.format(s_m))
```

100%|

```
| 10000/10000 [01:55<00:00, 86.93it/s]

There where 0 Singular Matrices
```

3.9 Results

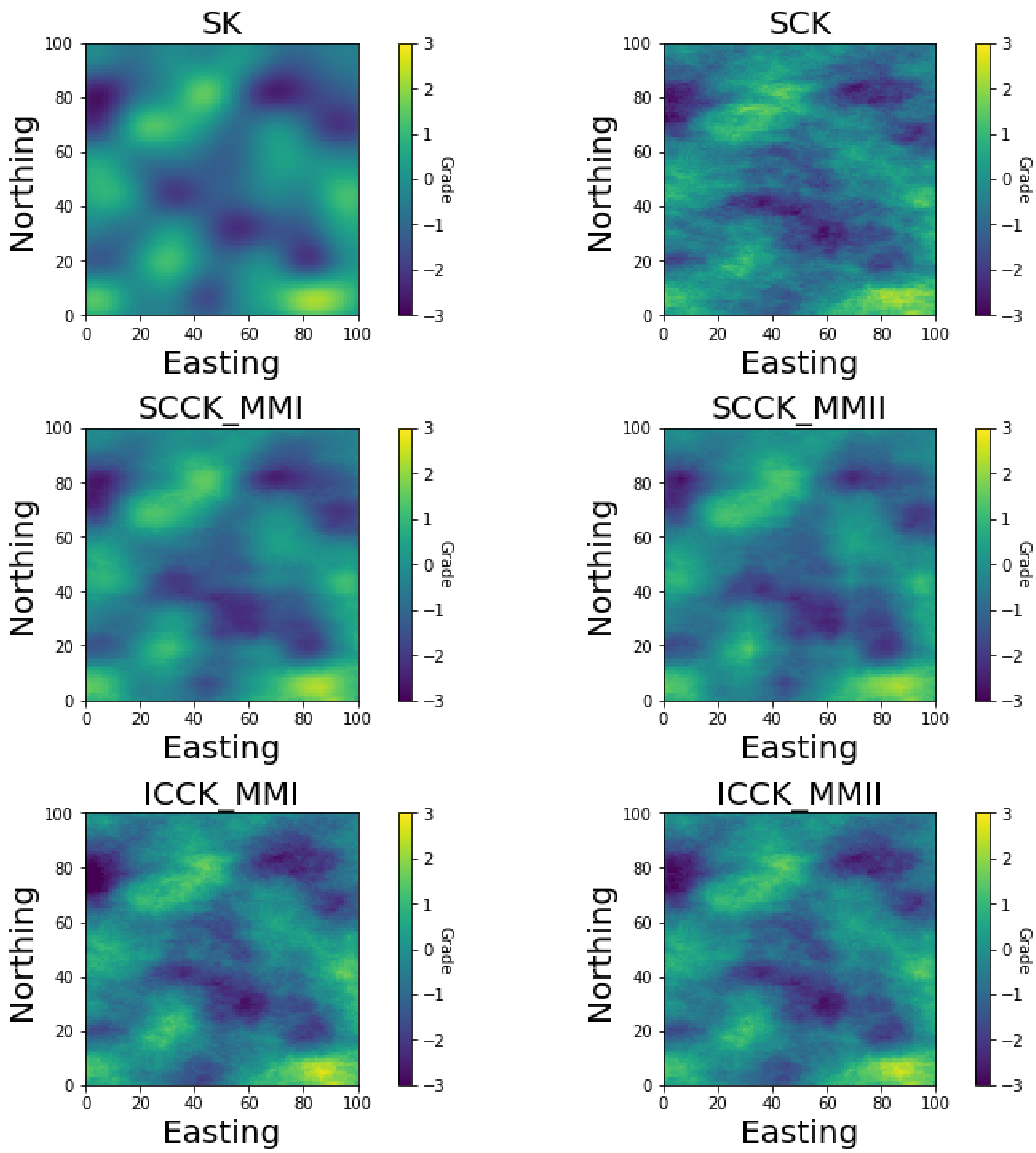
[36]:
```
#Setup Dictionary of Results
ktypes = ['SK','SCK','SCCK_MMI','SCCK_MMII','ICCK_MMI','ICCK_MMII']
k_est = [est_SK,est_Full_CCK,est_MM1,est_MM2,est_icck_MM1,est_icck_MM2]
ktypes_vals_dict = {}
j=0
for i in ktypes:
    ktypes_vals_dict[i] = {'Estimate':k_est[j],
                           'RMSE':np.
 ↪sqrt(mean_squared_error(k_est[j],truth['Primary'])),
                           'Mean':np.mean(k_est[j]),
                           'Variance':np.var(k_est[j])}
    j=j+1
```

3.9.1 Pixelplt

[37]:
```
#Setup Subplot Function
def plot_axe(figsize):
    #Set up Plotting Grid
    axes = []
    fig = plt.figure(constrained_layout=True,figsize=figsize)
    gs = gridspec.GridSpec(3, 4,figure=fig)
    gs.update(wspace=0.01,hspace=0.01)
    k=0
    for i in range (0,3):
        for j in range(1,3):
            k=k+1
            ax = locals()['ax{}'.format(k)] = plt.subplot(gs[i, (j-1)*2:j*2])
            axes.append(ax)
    return fig, gs, axes
```

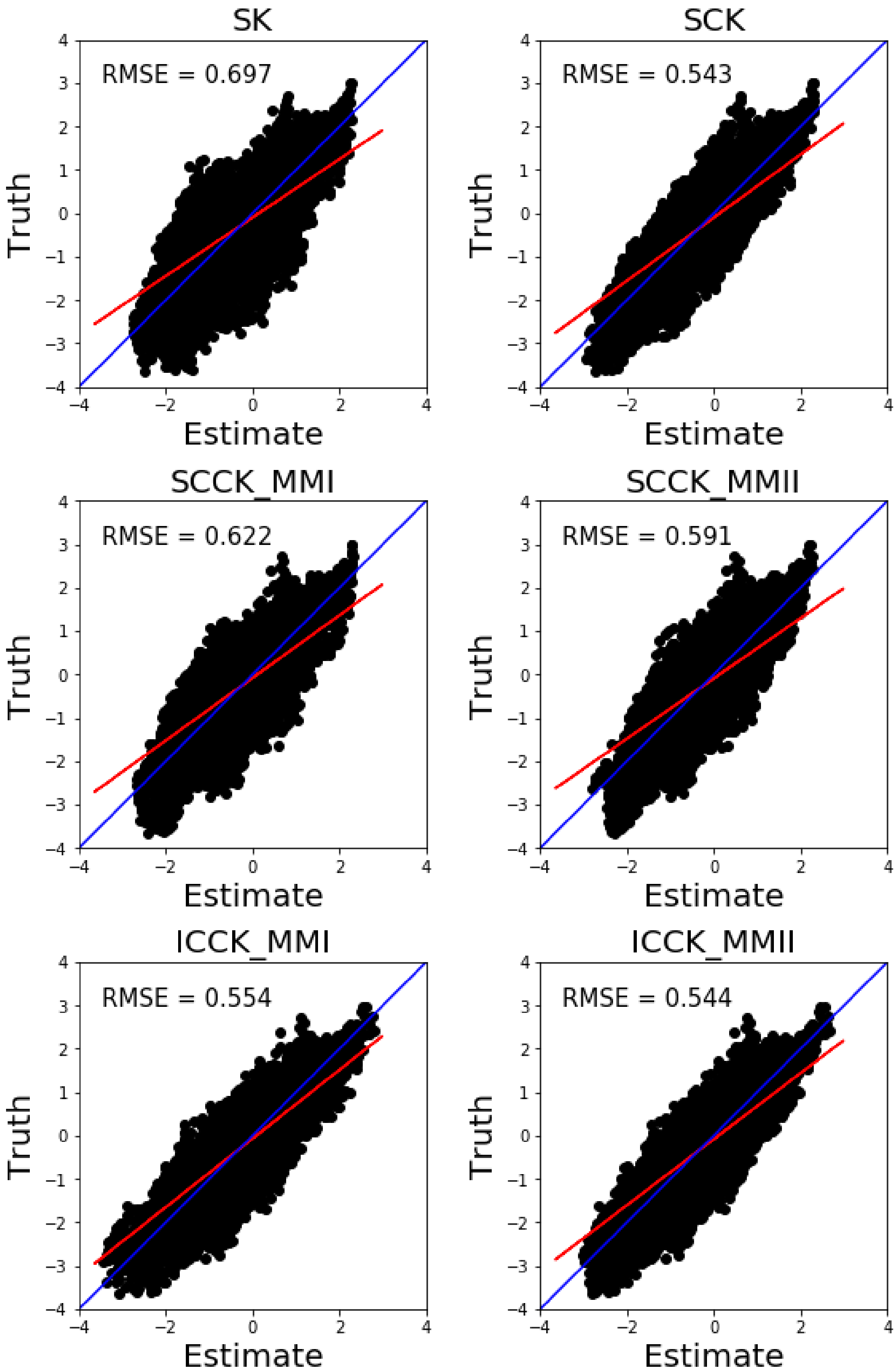
[38]:
```
fig, gs, axes = plot_axe(figsize=(10,10))
ax_i = 0
for i in ktypes:
    gridd['Estimate'] = ktypes_vals_dict[i]['Estimate']
    gridded = np.reshape(gridd.sort_values(by=['Y', 'X'], axis=0,␣
 ↪ascending=True)['Estimate'].values,
                         [100, 100], order='C')
    ax = axes[ax_i]
    plt_1 = ax.imshow(gridded, origin='lower', extent=[XMIN, XMAX, YMIN, YMAX],␣
 ↪aspect='equal',
```

```
                interpolation='none', vmin=SMIN, vmax=SMAX, cmap='viridis')
    ax.set_title('{}'.format(i),size=20)
    ax.set_xlabel('Easting',size=20)
    ax.set_ylabel('Northing',size=20)
    cbar = plt.colorbar(plt_1,ax=ax)
    cbar.set_label('Grade', rotation=270)
    ax_i = ax_i+1
plt.show()
fig.savefig('../0-Figures/estimates.png')
```



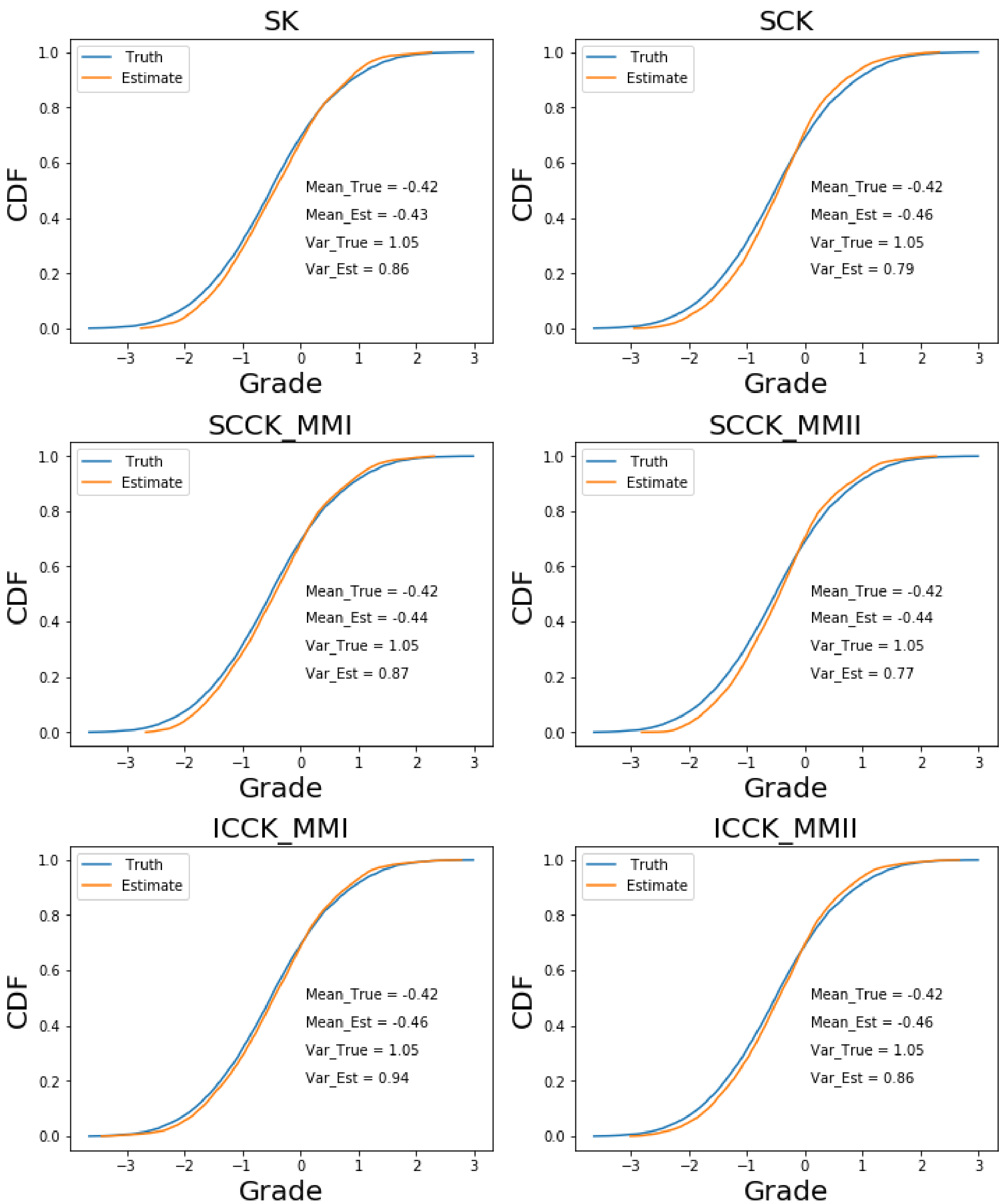
3.9.2 Data Reproduction

[39]:
```
fig, gs, axes = plot_axe(figsize=(8,12))
ax_i = 0
for i in ktypes:
    ax = axes[ax_i]
    ax.scatter(ktypes_vals_dict[i]['Estimate'],truth['Primary'],color='Black')
    ax.plot((truth['Primary']),
             np.poly1d(np.polyfit(truth['Primary'],␣
 ↪ktypes_vals_dict[i]['Estimate'], 1))((truth['Primary'])),
            color = 'Red')
    ax.set_title('{}'.format(i),size = 20)
    ax.set_xlim(-4,4)
    ax.set_ylim(-4,4)
    x_45 = np.linspace(*ax.get_xlim())
    ax.plot(x_45, x_45,color = 'blue')
    ax.set_xlabel('Estimate',size = 20)
    ax.set_ylabel('Truth',size = 20)
    ax.set_aspect('equal', 'box')
    ax.text(-3.5,3.0,'RMSE = {:.3f}'.
 ↪format(ktypes_vals_dict[i]['RMSE']),size=15)
    ax_i = ax_i + 1
plt.show()
fig.savefig('../0-Figures/Scatter_true.png')
```


SK
RMSE = 0.697
SCK
RMSE = 0.543
SCCK_MMI
RMSE = 0.622
SCCK_MMII
RMSE = 0.591
ICCK_MMI
RMSE = 0.554
ICCK_MMII
RMSE = 0.544
Truth
Estimate

3.9.3 Histogram Reproduction

[40]:
```
Mean_true = np.mean(datafl['Primary'])
var_true = np.var(datafl['Primary'])
fig, gs, axes = plot_axe(figsize=(10,12))
ax_i = 0
x_cdf_t, y_cdf_t = sorted(truth['Primary']), np.arange(len(truth['Primary'])) /␣
 ↪len(truth['Primary'])
for i in ktypes:
    ax = axes[ax_i]
    ax.plot(x_cdf_t, y_cdf_t,label = ' Truth')
    x_cdf, y_cdf = sorted(ktypes_vals_dict[i]['Estimate']),np.
 ↪arange(len(ktypes_vals_dict[i]['Estimate'])) /␣
 ↪len(ktypes_vals_dict[i]['Estimate'])
    ax.plot(x_cdf, y_cdf, label = 'Estimate')
    ax.set_title('{}'.format(i),size = 20)
    ax.set_xlabel('Grade',size = 20)
    ax.set_ylabel('CDF',size = 20)
    ax.legend(loc = 'best')
    ax.text(0.1,0.50,'Mean_True = {:.2f}'.format(Mean_true),size=10)
    ax.text(0.1,0.40,'Mean_Est = {:.2f}'.
 ↪format(ktypes_vals_dict[i]['Mean']),size=10)
    ax.text(0.1,0.30,'Var_True = {:.2f}'.format(var_true),size=10)
    ax.text(0.1,0.20,'Var_Est = {:.2f}'.
 ↪format(ktypes_vals_dict[i]['Variance']),size=10)
    ax_i = ax_i + 1
fig.savefig('../0-Figures/hist_rep.png')
```


SK
Truth
Estimate
CDF
Grade
Mean_True = -0.42
Mean_Est = -0.43
Var_True = 1.05
Var_Est = 0.86
SCK
Truth
Estimate
CDF
Grade
Mean_True = -0.42
Mean_Est = -0.46
Var_True = 1.05
Var_Est = 0.79
SCCK_MMI
Truth
Estimate
CDF
Grade
Mean_True = -0.42
Mean_Est = -0.44
Var_True = 1.05
Var_Est = 0.87
SCCK_MMII
Truth
Estimate
CDF
Grade
Mean_True = -0.42
Mean_Est = -0.44
Var_True = 1.05
Var_Est = 0.77
ICCK_MMI
Truth
Estimate
CDF
Grade
Mean_True = -0.42
Mean_Est = -0.46
Var_True = 1.05
Var_Est = 0.94
ICCK_MMII
Truth
Estimate
CDF
Grade
Mean_True = -0.42
Mean_Est = -0.46
Var_True = 1.05
Var_Est = 0.86

[ ]: