Kriging example

Since the global data base we used cannot be shared, we demonstrate using freely available data from Assumpcao et al. (2013) for South America, how the codes can be used.

For simplicity's sake we did not use two different categories here, but focused on the continental area instead, by simply discarding all points where the Moho depth is less than 30 km.

```
import numpy as np
import matplotlib.pyplot as plt

import clean_kriging
import sklearn.cluster as cluster

from func_dump import get_pairwise_geo_distance

import logging
logging.basicConfig(level=logging.DEBUG)

def test_cluster_size(point_data,max_size,do_plot=False,chosen_range=None,
        perc_levels=20):
    """Test effect of number of clusters on cluster radius and size
    """

    cluster_sizes = range(5,max_size,1)
    radius_1 = np.zeros((len(cluster_sizes),3))
    cluster_N = np.zeros((len(cluster_sizes),3))
    percentages = np.zeros((len(cluster_sizes),perc_levels+1))

    X = point_data
    Xsel = X
    pd = get_pairwise_geo_distance(Xsel[:,0],Xsel[:,1])

    for k,n_clusters in enumerate(cluster_sizes):
        model = cluster.AgglomerativeClustering(linkage='complete',
                                                affinity='precomputed',
                                                n_clusters=n_clusters)
        model.fit(pd)
        radius = np.zeros((n_clusters))
        cluster_members = np.zeros((n_clusters))
        for i,c in enumerate(np.unique(model.labels_)):
            ix = np.where(model.labels_==c)[0]
            radius[i] = 0.5*pd[np.ix_(ix,ix)].max()
            cluster_members[i] = np.sum(model.labels_==c)
        r1i,r1a,r1s = (radius.min(),radius.max(),radius.std())
        radius_1[k,0] = r1i
        radius_1[k,1] = r1a
        radius_1[k,2] = np.median(radius)
        percentages[k,:] = np.percentile(radius,np.linspace(0,100,perc_levels+1))

    radius_1 = radius_1*110.0
    percentages = percentages*110.0

    if do_plot:
        plt.plot(cluster_sizes,radius_1)
```

```
        for i in range(perc_levels):
            if i<perc_levels/2:
                alpha = (i+1)*2.0/perc_levels
            else:
                alpha = (perc_levels-i)*2.0/perc_levels
            plt.fill_between(cluster_sizes,percentages[:,i],percentages[:,i+1],
                alpha=alpha,facecolor='green',edgecolor='none')
    if not chosen_range is None:
        return cluster_sizes[np.argmin(np.abs(radius_1[:,2]-chosen_range))]


def cluster_map(krigor):
    """Visualize distribution spatial distribution of a cluster
    """
    fig = plt.figure(figsize=(7,11))

    Xsel = krigor.X

    model = krigor.cluster_results[0]
    n_clusters = model.n_clusters
    cmap = plt.cm.get_cmap("jet",n_clusters)

    clu = model.cluster_centers_
    pointsize = np.sqrt(np.bincount(model.labels_))

    for i in range(len(Xsel)):
        j = model.labels_[i]
        if (Xsel[i,0]*clu[j,0])<0 and np.abs(np.abs(clu[j,0])-180.0) < 10.0:
            continue
        plt.plot((Xsel[i,0],clu[j,0]),(Xsel[i,1],clu[j,1]),
                 color=cmap(model.labels_[i]),alpha=0.5)

    print clu.shape,n_clusters,pointsize.shape

    plt.scatter(clu[:,0],clu[:,1],7.5*pointsize,np.linspace(0,n_clusters,n_clusters),'s',
                    alpha=1.0,cmap=cmap,edgecolor='r',linewidth=1.5)

    plt.scatter(Xsel[:,0],Xsel[:,1],2,model.labels_,cmap=cmap,alpha=1.0,edgecolor='k')
    plt.axis('equal')
    plt.xlabel('Longitude')
    plt.ylabel('Latitude')
    #plt.xlim([-90,-20])
```

Data input

We load the file shipped together with this example. See the inside of the files for references to the sources.

```
point_data = np.loadtxt("Seismic_Moho_Assumpcao.txt",delimiter=",")
point_data[:,2] = -0.001*point_data[:,2]

point_data = point_data[point_data[:,2]>30.0,:]

lon = np.arange(np.round(point_data[:,0].min()),np.round(point_data[:,0].max()+1),1)
```

```
lat = np.arange(np.round(point_data[:,1].min()),np.round(point_data[:,1].max()+1),1)

lonGrid,latGrid = np.meshgrid(lon,lat)
```

Prior specification

We want to use inverse gamma priors for nugget, sill and range. The inverse gamma distribution is defined in terms of the parameters α and β, which we derive here from a specified mean and variance.

$$\mu = \text{Mean} = \frac{\beta}{\alpha - 1} \quad \text{and} \quad \sigma^2 = \text{var} = \frac{\beta^2}{(\alpha-1)^2(\alpha-2)}$$

Thus,

$$\alpha = 2 + \frac{\mu^2}{\sigma^2} \quad \text{and} \quad \beta = \frac{\mu^3}{\sigma^2} + \mu$$

The variable `moments` contains mean and variance for all nugget, sill and range. The last dimension of `moments` would be used, if there are different categories (i.e. ocean vs. continent), but in this example this is not required.

```
moments = np.zeros((3,2,1))
moments[:,:,0] = np.array(((1.0,3.0**2),(40.0,40.0**2),(10.0,10.0**2)))
beta = moments[:,0,:]**3/moments[:,1,:]+moments[:,0,:]
alpha = 2 + moments[:,0,:]**2 / moments[:,1,:]
```

Clustering

All important routines are contained in objects of the class `MLEKrigor`. Such an object is created by passing it longitude,latitude,value and category. In this example, all category values are simply zero. Any clustering algorithm from the scikit-learn package can be used. Any options contained in the dictionary `clusterOption` will be passed to the constructor.

After clustering, the covariance parameters for all clusters are determined (`krigor._fit_all_clusters`).

```
cat = np.ones((point_data.shape[0]),dtype=int)

krigor = clean_kriging.MLEKrigor(point_data[:,0],point_data[:,1],point_data[:,2],cat)
clusterOptions=[{'linkage':'complete','affinity':'precomputed','n_clusters':16}]

krigor._cluster_points(cluster.AgglomerativeClustering,options=clusterOptions,use_pd=True)
krigor._detect_dupes()
krigor._fit_all_clusters(minNugget=0.5,minSill=1.0,
    hyperpars=np.dstack((alpha,beta)),prior="inv_gamma",maxRange=None)


krigDict = {"threshold":1,"lambda_w":1.0,"minSill":1.0,
             "minNugget":0.5,
            "maxAbsError":4.0,"maxRelError":2.0,"badPoints":None,
            "hyperPars":np.dstack((alpha,beta)),"prior":"inv_gamma",
            "blocks":10}
```

```
clean_kriging.py:119: RuntimeWarning: divide by zero encountered in log
  return np.sum(-(hyperpars[:,0]+1)*np.log(vals) - hyperpars[:,1]/vals)
clean_kriging.py:119: RuntimeWarning: divide by zero encountered in divide
  return np.sum(-(hyperpars[:,0]+1)*np.log(vals) - hyperpars[:,1]/vals)
clean_kriging.py:119: RuntimeWarning: invalid value encountered in subtract
  return np.sum(-(hyperpars[:,0]+1)*np.log(vals) - hyperpars[:,1]/vals)
```

```
cluster_map(krigor)
```

```
(16L, 2L) 16 (16L,)
```

Outlier detection

This is the most time-consuming step. The routine `jacknife` performs the hold-one-out cross validation to detect possible outliers. Two criteria are used to determine if a point is an outlier.

1. The **absolute** prediction error needs to be 4 km or more.
2. The prediction error is twice as high as the estimated error.

This is controlled by the variables `maxAbsErr` and `maxRelErr` passed to the function `jacknife`. The third parameter (λ_w) controls how the covariance parameters are interpolated.

There are two rounds of outlier detection (see main text for explanation).

```
sigma1,new_chosen = krigor.jacknife(4.0,2.0,100.0)
krigor.chosen_points = new_chosen.copy()
krigor._fit_all_clusters(minNugget=0.5,minSill=1.0,
        hyperpars=krigDict["hyperPars"],prior="inv_gamma",maxRange=None)

sigma2,new_new_chosen = krigor.jacknife(4.0,2.0,100.0)
krigor.chosen_points = new_new_chosen.copy()
krigor._fit_all_clusters(minNugget=0.5,minSill=1.0,
        hyperpars=krigDict["hyperPars"],prior="inv_gamma",maxRange=None)
```

```
clean_kriging.py:119: RuntimeWarning: divide by zero encountered in log
  return np.sum(-(hyperpars[:,0]+1)*np.log(vals) - hyperpars[:,1]/vals)
clean_kriging.py:119: RuntimeWarning: divide by zero encountered in divide
  return np.sum(-(hyperpars[:,0]+1)*np.log(vals) - hyperpars[:,1]/vals)
clean_kriging.py:119: RuntimeWarning: invalid value encountered in subtract
  return np.sum(-(hyperpars[:,0]+1)*np.log(vals) - hyperpars[:,1]/vals)
INFO:root:Jacknife category 0 label 1
...
```

Interpolation

To run the actual interpolation, the `predict` method of the `MLEKrigor` is used. It takes, longitude, latitude and category as main input. In addition, λ_w needs to be specified. This mainly affects the obtained uncertainties. If desired, the full covariance matrix can also be calculated, but due to memory constraints, by default only the variance (main diagonal) is computed.

Note that `predict` does not respect the shape of the input points and the outputs needs to be reshaped. Furtheremore, the **variance** of the error is returned (to be compatible with the full covariance case) not the standard deviation!

```
cat_grid = np.ones(lonGrid.shape,dtype=int)

pred,krigvar,predPars = krigor.predict(lonGrid.flatten(),latGrid.flatten(),
```

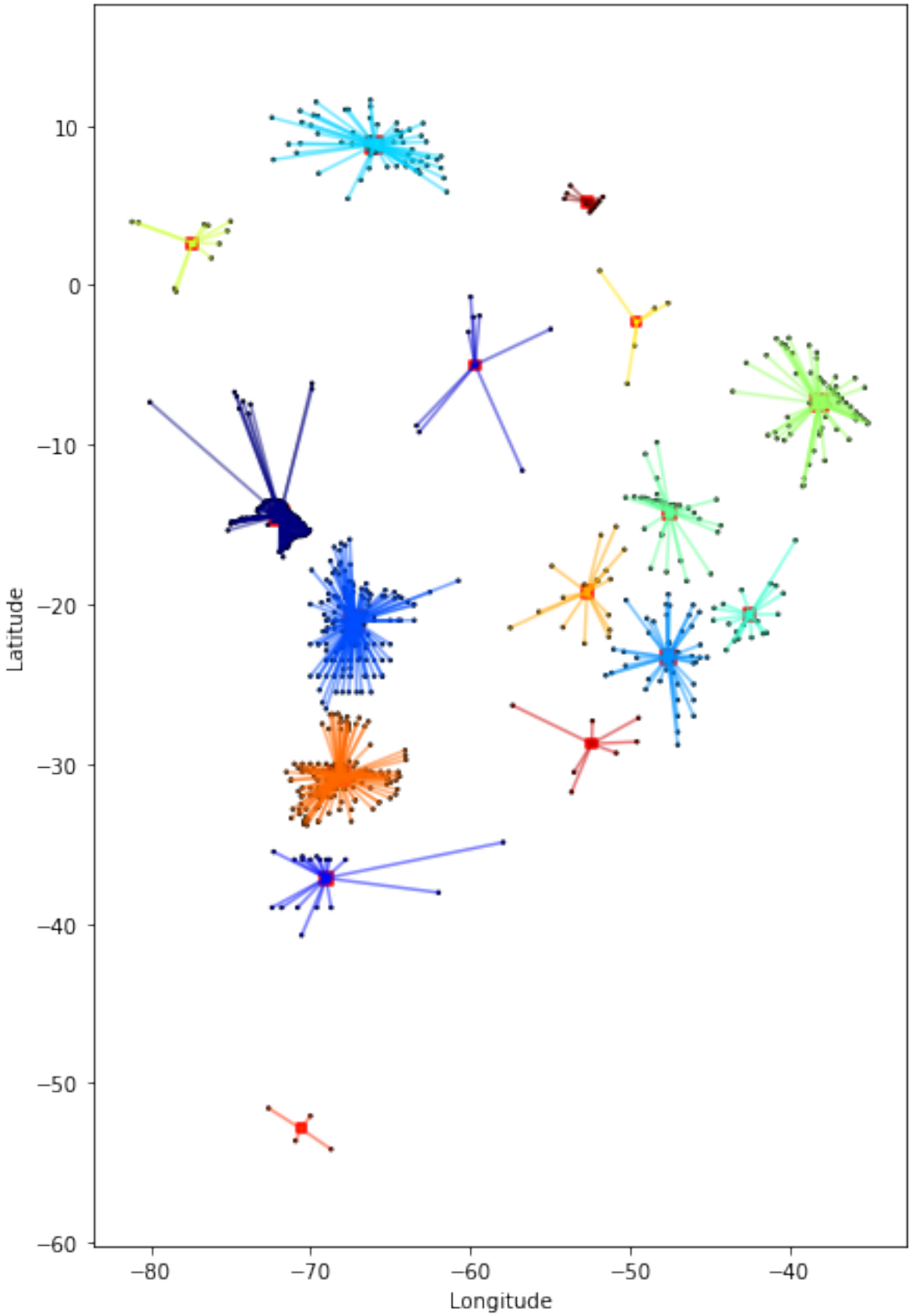

Figure 1: In this map, the individual points are connected with lines to their respective cluster center

```
                                    cat_grid.flatten(),lambda_w=100.0,
                                    get_covar=False)

pred = pred.reshape(lonGrid.shape)
krigvar = krigvar.reshape(lonGrid.shape)
```

```
Solving kriging system for category 1 with no. points 3149 699
```

```
plt.figure()
plt.contourf(lonGrid,latGrid,pred)
cbar = plt.colorbar()
cbar.set_label('Moho depth [km]')
plt.axis('equal')
plt.xlabel('Longitude')
plt.ylabel('Latitude')

plt.figure()
plt.contourf(lonGrid,latGrid,np.sqrt(krigvar))
cbar = plt.colorbar()
cbar.set_label('Moho uncertainty [km]')
plt.axis('equal')
```

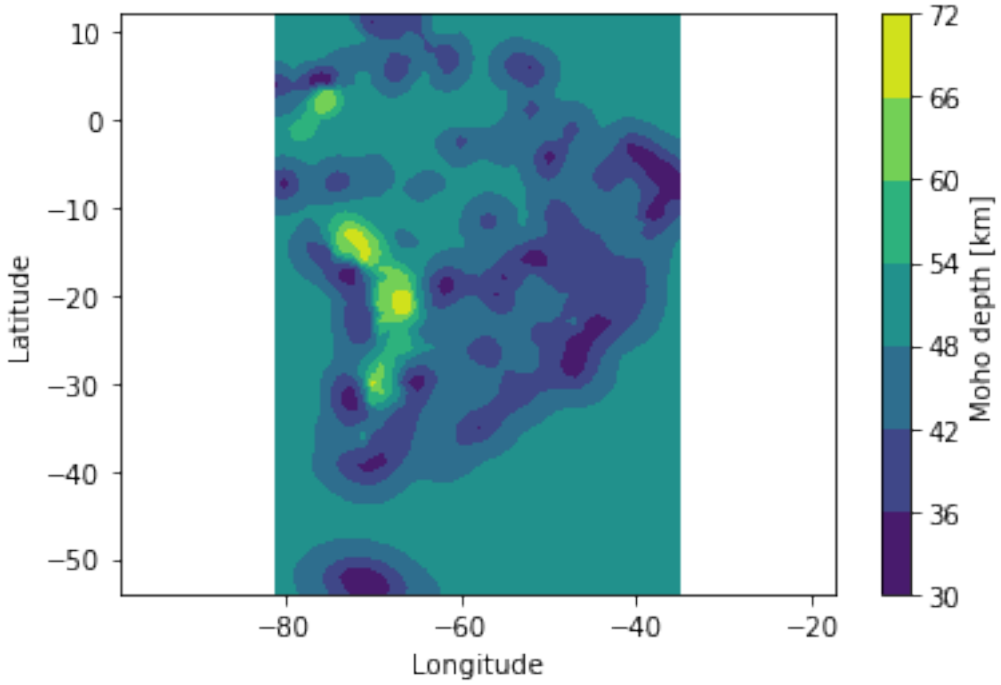

Figure 2: Interpolated Moho depth

Note that in this case, one should not interpret results in the oceanic domain, since they were excluded from the interpolation.

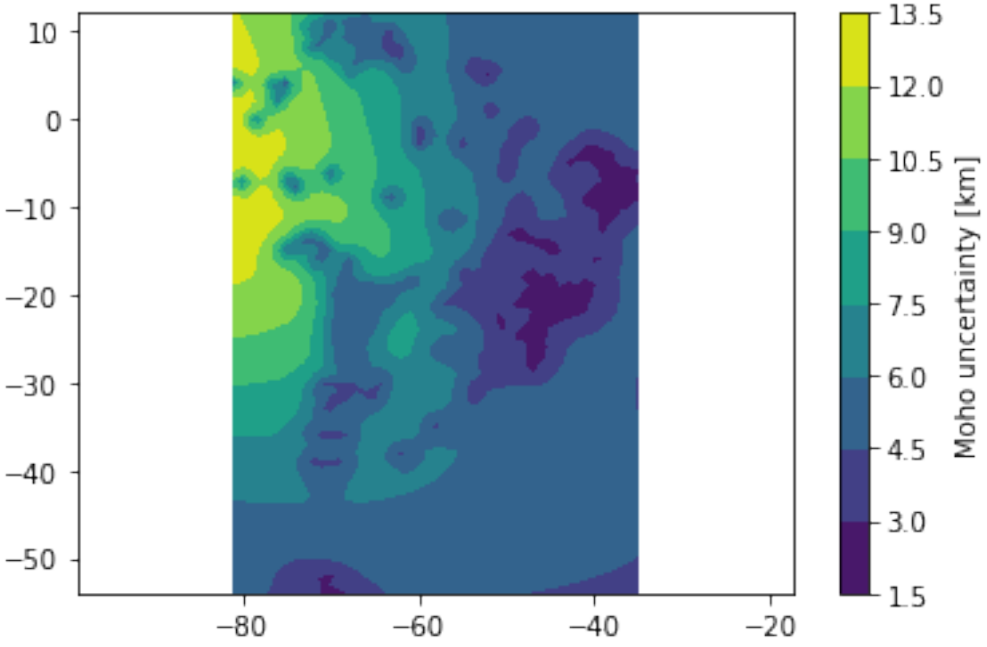

Figure 3: Interpolated Moho depth unertainty