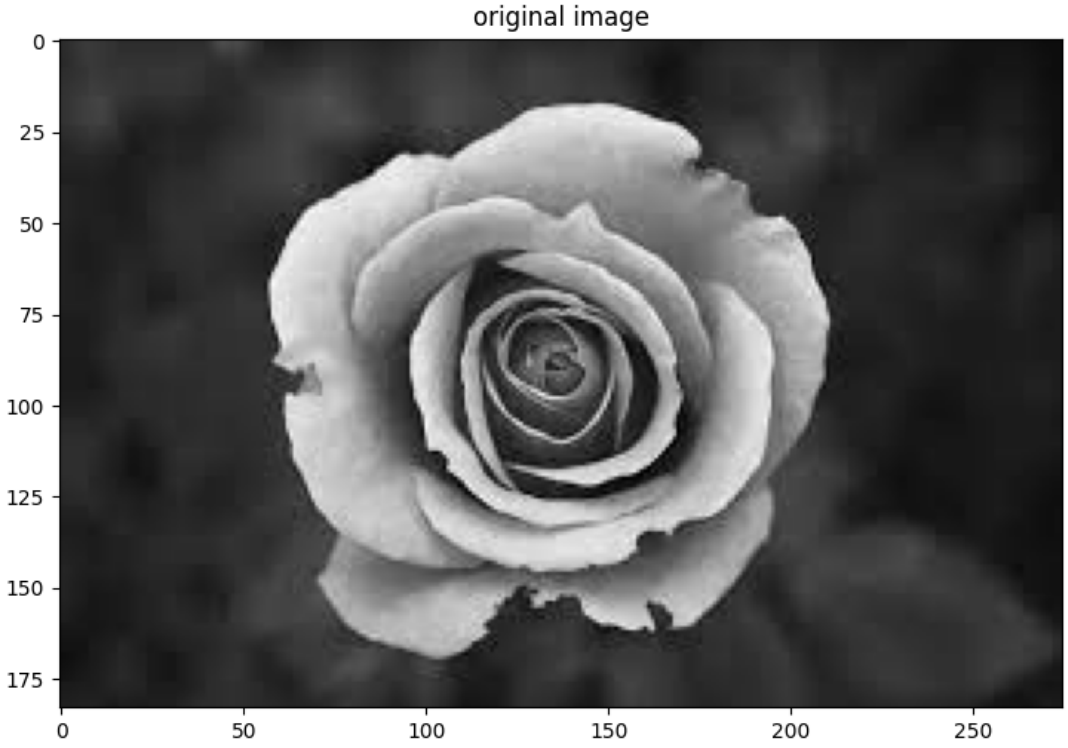
پروژه تبدیلات

علیرضاابراهیمی
۸۱۰۳۰۱۰۱۷

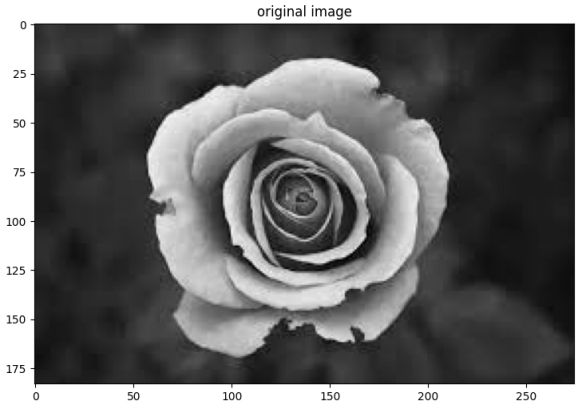
قسمت اول :

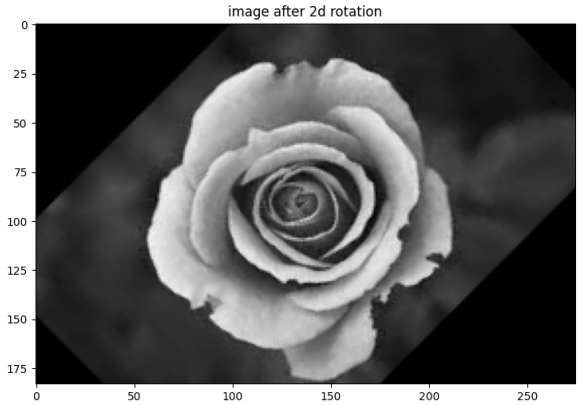
در این مرحله هدف اعمال تغییرات scale ، rotate ، shift و shear است . از عکس زیر به عنوان ورودی استفاده می‌شود .



برای اعمال تغییرات بالا از پکیج open-cv استفاده شده است .
: 2d rotation

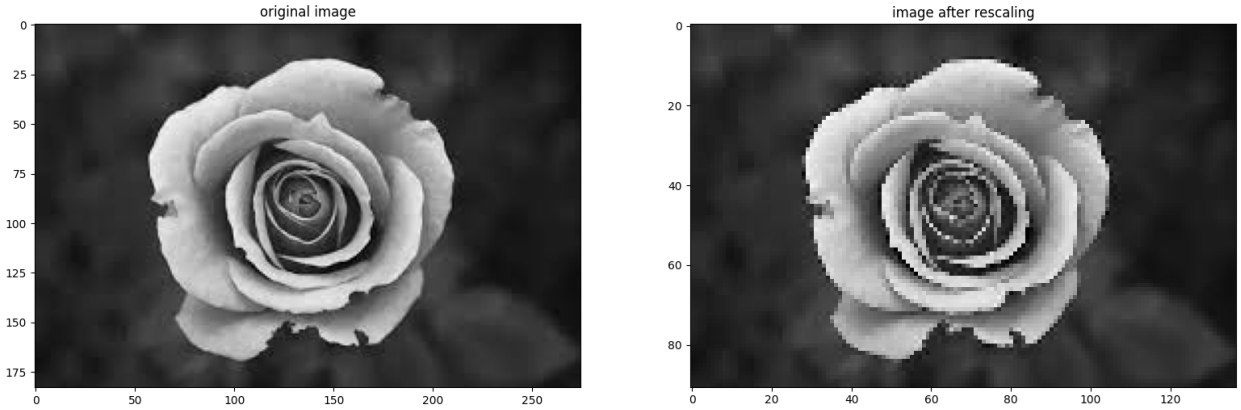
بعد از اعمال دوران ۴۵ درجه عکس حاصل به صورت زیر است .





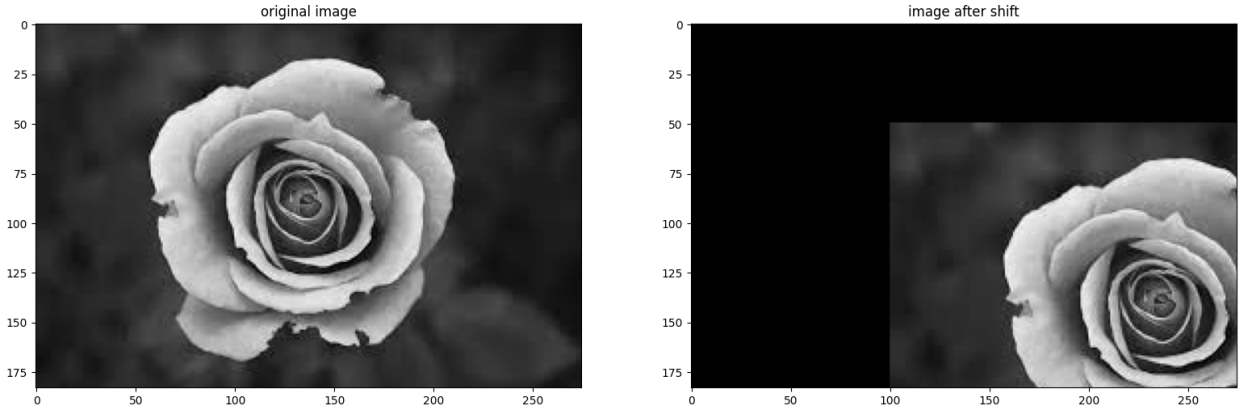
: Pixel scaling

با انجام عملیات reshape میزان resolution مکانی را تغییر می‌دهیم که نتیجه به صورت زیر است .



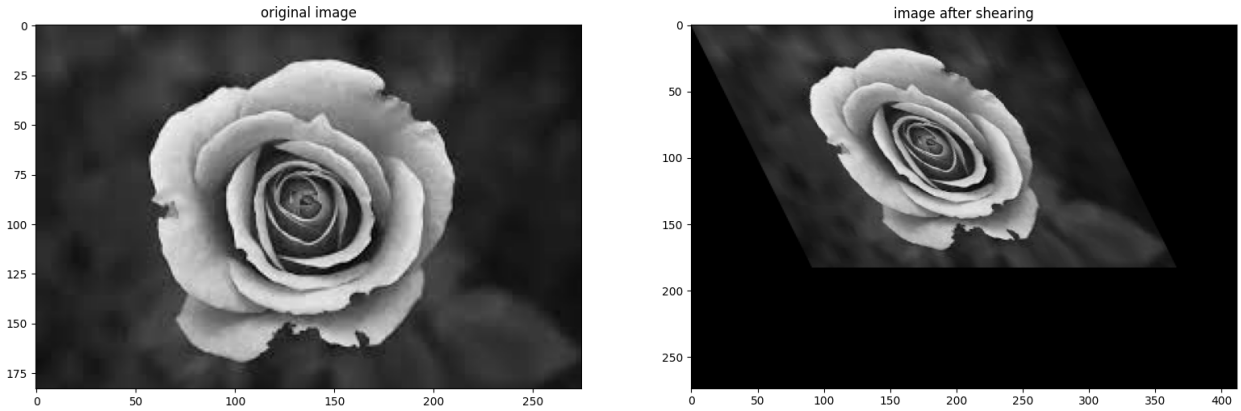
: Shift

در این عملیات تمامی نقاط عکسی با دو شیفت در جهت x و y به نقاط جدید منتقل می‌شوند .



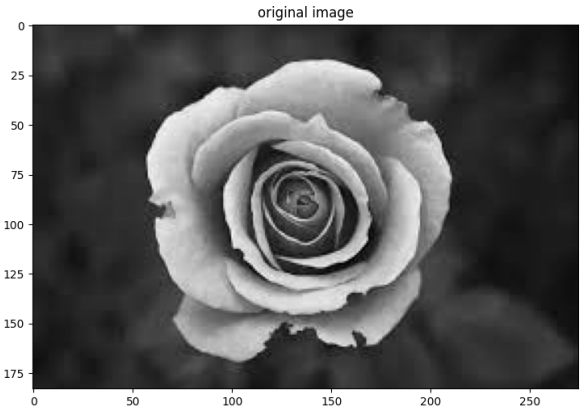
: Shear

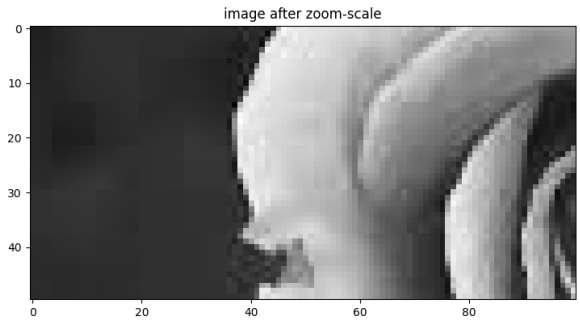
این عملیات به منظور ایجاد دوران در فضای سوم ایجاد می‌گردد .



: Scale-zoom

در این عملیات اندازه pixelها تغییر نمی‌کند تنها نمایش تصویر با بزرگنمایی صورت می‌گیرد .



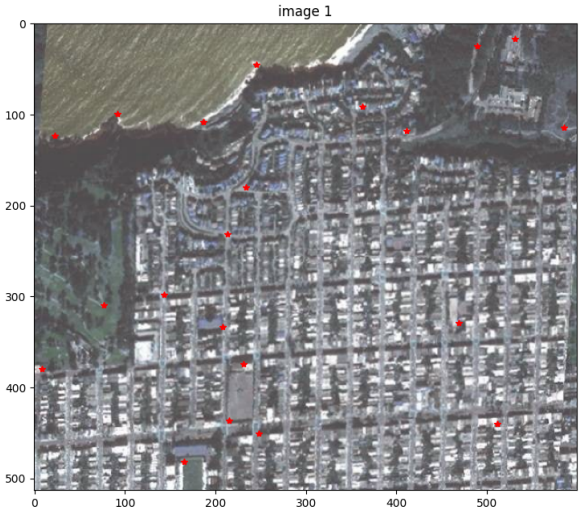


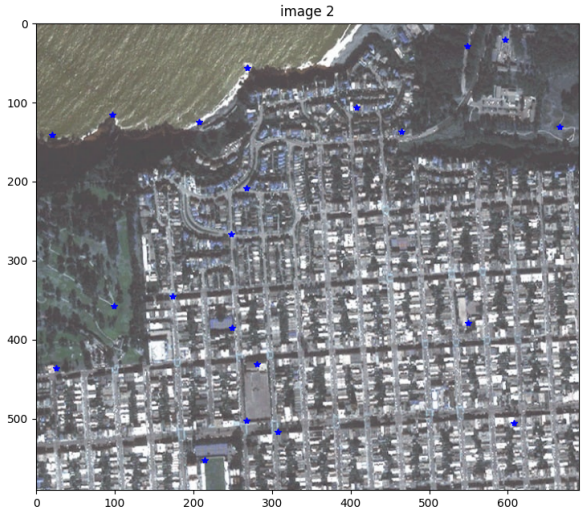

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import imutils
FILE_NAME = 'download.jpeg'
img = cv2.imread(FILE_NAME)
(rows, cols) = img.shape[:2]
(height, width) = img.shape[:2]
######################################### rotation
#########################################


M = cv2.getRotationMatrix2D((cols / 2, rows / 2), 45, 1)
# res = cv2.warpAffine(img, M, (cols, rows))
res = imutils.rotate(img , 45)
plt.subplot(121)
plt.imshow(img)
plt.title('original image')

plt.subplot(122)
plt.imshow(res)
plt.title('image after 2d rotation')
plt.show()
######################################### rescale
#########################################

res = cv2.resize(img, (int(width / 2), int(height / 2)), interpolation=cv2.INTER_CUBIC)
plt.subplot(121)
plt.imshow(img)
plt.title('original image')
plt.subplot(122)
plt.imshow(res)
```

```
plt.title('image after rescaling')
plt.show()
########################################## shift
##########################################

M = np.float32([[1, 0, 100], [0, 1, 50]])
res = cv2.warpAffine(img, M, (cols, rows))
plt.subplot(121)
plt.imshow(img)
plt.title('original image')
plt.subplot(122)
plt.imshow(res)
plt.title('image after shift')
plt.show()
########################################## shear
##########################################

plt.subplot(121)
plt.imshow(img)
plt.title('original image')
M = np.float32([[1, 0.5, 0],
              [0, 1  , 0],
              [0, 0  , 1]])
sheared_img = cv2.warpPerspective(img,M,(int(cols*1.5),int(rows*1.5)))
plt.subplot(122)
plt.imshow(sheared_img)
plt.title('image after shearing')
plt.show()
########################################## scale-zoom
##########################################

res = np.array(img)[50:100 , 20:120]
plt.subplot(121)
plt.imshow(img)
plt.title('original image')
plt.subplot(122)
plt.imshow(res)
plt.title('image after zoom-scale')
plt.show()
```

قسمت دوم :

در این قسمت هدف ایجاد یک معادله برای انتقال مختصات پیکسلی نقاط در یک عکس به مختصات زمینی نقاط متناظر در عکس دوم است که برای این منظور از معادلات affine ، quadratic و cubic استفاده می‌شود .
ابتدا به صورت دستی تعدادی نقطه متناظر از عکس اول و دوم انتخاب می‌کنیم که محل آن‌ها در دو عکس به صورت زیر است .





بعد از بدست آوردن مختصات پیکسلی نقاط در عکس دوم ، مختصات زمینی آن را با استفاده از رابطه زیر محاسبه می‌کنیم .

$$X_{earth} = X_{top-left} + x_{pixel} \times d_x$$
$$Y_{earth} = X_{top-left} - y_{pixel} \times d_y$$

در مرحله بعد از با استفاده از معادلات زیر و کمترین مربعات ، پارامترهای تبدیل مختصات را بدست می‌آوریم .

: Affine

$$X = a0 + a1x + a2y$$
$$Y = a3 + a4x + a5y$$

: Quadratic

$$X = a0 + a1x + a2y + a3xy + a4x^2 + a5y^2$$
$$Y = a6 + a7x + a8y + a9xy + a10x^2 + a11y^2$$

: Cubic

$$X = a0 + a1x + a2y + a3xy + a4x^2 + a5y^2 + a6x^2y + a7xy^2 + a8x^3 + a9y^3$$
$$X = a10 + a11x + a12y + a13xy + a14x^2 + a15y^2 + a16x^2y + a17xy^2 + a18x^3 + a19y^3$$

مقدار خطا برای هر پیکسل و کل پیکسل‌ها در دیتاست‌های train و test با استفاده از روش‌های متفاوت به صورت زیر است :

دیتاست train :

تغییرات پیکسل‌ها در روش affine	تغییرات پیکسل‌ها در روش quadratic	تغییرات پیکسل‌ها در روش cubic
[[-0.13112249 -1.20433831] [-0.23306891 0.88625144] [-0.74371793 0.01150903] [-0.23280758 6.08861237] [1.54649768 -0.63809804] [2.86720816 1.0499547] [-0.6016412 -1.13892803] [-1.55908988 -0.68721086] [1.81634854 0.94840265] [-0.65924536 0.71178463] [-2.50558044 2.87009219] [-1.74782551 0.82667776] [0.66917635 -4.08794237] [-0.64100519 -1.73370024] [1.05939103 0.3091049] [1.59284225 -0.92971109] [-0.49635952 -3.28246076]]	[[1.15391944 0.44242132] [-0.05016541 -0.27015639] [-0.77219755 -1.49162173] [-0.81380297 4.64863962] [0.79425039 0.06066617] [2.96344324 -0.25056654] [-0.09530908 1.06412258] [-0.67519464 0.87823929] [1.01139525 0.06589676] [-0.16443773 0.10286832] [-1.80376884 3.24184824] [-1.76112248 -0.76952552] [0.01288373 -1.47415006] [-0.2162638 -1.90484588] [-0.86204581 0.35713316] [1.15428673 -1.94077195] [0.12412949 -2.7601961]]	[[0.48229174 -0.55203255] [0.47070606 0.14543302] [-0.64002483 -0.45717838] [-0.60262722 1.64173415] [0.66335417 -0.03520819] [1.84229444 1.58078237] [-0.5861586 0.1954983] [0.18027433 1.30501286] [0.27963157 -0.94857843] [0.20595256 -0.20379333] [-0.41160644 0.7971856] [-2.40514989 0.16573031] [-0.57005859 0.22542378] [-0.74414073 0.67611993] [0.00782994 0.27371359] [1.33317292 -2.21301311] [0.49425655 -2.59685064]]

دقت کلی روش affine	دقت کلی روش quadratic	دقت کلی روش cubic
2.2178817138615097	1.7671223783969743	1.2055413982998213

دیتاست test :

تغییرات پیکسل‌ها در روش affine	تغییرات پیکسل‌ها در روش quadratic	تغییرات پیکسل‌ها در روش cubic
[[-0.74309792 -1.70605128] [5.13982586 2.78539206] [-2.41277627 1.32093167] [-1.03378863 2.37722111]]	[[1.29917665 3.38986483] [5.68150968 2.20965414] [-1.35412505 3.42599188] [-0.32094605 2.55156962]]	[[3.04121337 -8.01830602] [6.43447732 1.14800246] [0.46921797 -1.41413021] [-0.41911051 2.87093813]]

دقت کلی روش affine	دقت کلی روش quadratic	دقت کلی روش cubic
3.262470301147908	3.9954847511083686	4.875767992105533

تحلیل خطا :

با توجه به مقادیر کلی خطا می‌توان دید که استفاده کرد از درجات بالاتر معادله دقت بهتری را برای دیتای آموزش به دست می‌آورد اما در دیتای تست این ترتیب برعکس است و نشان می‌دهد هرچه از معادله با درجات بالاتر استفاده کنیم دچار خطای overfit می‌شویم .

کد بخش اصلی :

```
import matplotlib.pyplot as plt
import numpy as np
import cv2
import random
from matching_functions import match , coord_transform

img2 = cv2.imread('photo_1.jpg')
img1 = cv2.imread('photo_2.jpg')
tifpath = '/home/alireza/Desktop/seg/QuickBird_1.tif'
q = input('to determine points press 1 and to use predetermined points press 2 :  ')
q = int(q)


if q == 1:
   mutable_object = {}
   fig = plt.figure()
   coords = []
   def onclick(event):
       X_coordinate = int(event.xdata)
       Y_coordinate = int(event.ydata)
       coord = [X_coordinate , Y_coordinate]
       coords.append(coord)
       counter = len(coords)
       C = np.array(coords)
       if counter % 2 ==0 :
           print(f'coordsinate of point in image one : {C[-1 , :]} / coordinates of point in
image two : {C[-2 , :]}')
       else:
           point_number = int(counter / 2)
           print(f'**************  point number {point_number}  **************')
           print('point of image on have been selected')
   cid = fig.canvas.mpl_connect('button_press_event', onclick)
```

```
    plt.subplot(121)
    plt.imshow(img1)
    plt.title('image 1')

    plt.subplot(122)
    plt.imshow(img2)
    plt.title('image 2')

    plt.show()
    coords = np.array(coords)
    coords1 = np.empty((round(coords.shape[0] / 2) , 2))
    coords2 = np.empty((round(coords.shape[0] / 2) , 2))

    m = 0
    n = 0
    for i in range(len(coords)):
        if i % 2 == 0:
            coords1[m , :] = coords[i , :]
            m = m + 1
        else :
            coords2[n, :] = coords[i, :]
            n = n + 1

    coords1 = np.array(coords1)
    coords2 = np.array(coords2)

    np.savetxt('file1.txt', coords1)
    np.savetxt('file2.txt', coords2)

else:
    coords1 = np.loadtxt('file1.txt')
    coords2 = np.loadtxt('file2.txt')



plt.subplot(121)
plt.imshow(img1)
plt.plot(coords1[: , 0] , coords1[: , 1] , 'r*')
plt.title('image 1')

plt.subplot(122)
plt.imshow(img2)
plt.title('image 2')
plt.plot(coords2[: , 0] , coords2[: , 1] , 'b*')


rate = 0.2
random.seed(1234)
k = round(rate * coords1.shape[0])
test_ind = random.sample(range(0 , coords1.shape[0]) , k)

# transforming coordinates from pixel to earth

coords2 = coord_transform(coords2 , tifpath)
```

```
# creating test and train dataset     e = entry / t = target

test_e = coords1[test_ind , :]
test_t = coords2[test_ind , :]
train_e = np.delete(coords1 , test_ind , axis = 0)
train_t = np.delete(coords2 , test_ind , axis = 0)

func = 'affine'  #can be quadratic or cubic or affine
match(train_e , train_t , test_e , test_t , method = func)

plt.show()
```

کد توابع :

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as ss
import rasterio as rio

def rmse(res):
   dif_vector =np.sqrt(res[: , 0] **2 + res[: , 1] **2)
   rmse = np.mean(dif_vector)
   return rmse

def match(train_e , train_t , test_e , test_t , method = 'quadric'):
   x = train_e
   y = train_t
   X = test_e
   Y = test_t
   y_label = np.hstack((y[: , 0] , y[: , 1]))
   Y_label = np.hstack((Y[: , 0] , Y[: , 1]))

   if method == 'affine' :
       # X = a0 + a1*x + a2*y
       B1 = np.column_stack((np.ones([len(x[: , 0]), 1]) , np.array([x[: , 0],x[: , 1]]).T ,
np.zeros([len(x), 3])))
       B2 = np.column_stack((np.zeros([len(x), 3]), np.ones([len(x[: , 0]), 1])
,np.array([x[: , 0], x[: , 1]]).T))
       A = np.vstack((B1, B2))
       Params = np.linalg.inv(A.T @ A) @ (A.T @ y_label)

       xnew_train = Params[0] + Params[1] * x[:, 0] + Params[2] * x[:, 1]
       ynew_train = Params[3] + Params[4] * x[:, 0] + Params[5] * x[:, 1]

       xnew_test = Params[0] + Params[1] * X[: , 0] + Params[2] * X[: , 1]
       ynew_test = Params[3] + Params[4] * X[: , 0] + Params[5] * X[: , 1]

       res_image_train = np.column_stack((xnew_train, ynew_train)) - y
       res_image_test = np.column_stack((xnew_test, ynew_test)) - Y

       print('********************* train_res *********************')
       print(res_image_train)
       print('********************* test_res *********************')
```

```
        print(res_image_test)
        print('********************** train_rmse **********************')
        rmse_train = rmse(res_image_train)
        print(rmse_train)
        print('********************** test_rmse **********************')
        rmse_test = rmse(res_image_test)
        print(rmse_test)

    if method == 'quadratic' :
        # X = a0 + a1*x + a2*y + a3*x**2 + a4*y**2 + a5*x*y
        B1 = np.column_stack((np.ones([len(x[: , 0]), 1]) , np.array([x[: , 0],x[: , 1],x[: ,
0]**2 , x[: , 1]**2 , x[: , 0]*x[: , 1]]).T , np.zeros([len(x), 6])))
        B2 = np.column_stack((np.zeros([len(x), 6]), np.ones([len(x[: , 0]), 1])
,np.array([x[: , 0], x[: , 1],x[: , 0]**2 , x[: , 1]**2 ,x[: , 0] * x[: , 1]]).T))
        A = np.vstack((B1, B2))
        Params = np.linalg.inv(A.T @ A) @ (A.T @ y_label)

        xnew_train = Params[0] + Params[1] * x[:, 0] + Params[2] * x[:, 1] + Params[3] * x[:,
0] ** 2 + Params[4] * x[:,1] ** 2 + Params[5] * x[:, 0] * x[:, 1]
        ynew_train = Params[6] + Params[7] * x[:, 0] + Params[8] * x[:, 1] + Params[9] * x[:,
0] ** 2 + Params[10] * x[:,1] ** 2 + Params[11] * x[:, 0] * x[:, 1]

        xnew_test = Params[0] + Params[1] * X[: , 0] + Params[2] * X[: , 1] + Params[3] * X[:
, 0] ** 2 + Params[4] * X[: , 1]**2 + Params[5] * X[: , 0] * X[: , 1]
        ynew_test = Params[6] + Params[7] * X[: , 0] + Params[8] * X[: , 1] + Params[9] * X[:
, 0] ** 2 + Params[10] * X[: , 1] ** 2 + Params[11] * X[:, 0] * X[:, 1]

        res_image_train = np.column_stack((xnew_train, ynew_train)) - y
        res_image_test = np.column_stack((xnew_test, ynew_test)) - Y

        print('********************** train_res **********************')
        print(res_image_train)
        print('********************** test_res **********************')
        print(res_image_test)
        print('********************** train_rmse **********************')
        rmse_train = rmse(res_image_train)
        print(rmse_train)
        print('********************** test_rmse **********************')
        rmse_test = rmse(res_image_test)
        print(rmse_test)

    if method == 'cubic' :
        # X = a0 + a1*x + a2*y + a3*x**2 + a4*y**2 + a5*x*y + a6*x**2*y + a7*x*y**2 + a8*x**3
+ a9*y**3
        B1 = np.column_stack((np.ones([len(x[:, 0]), 1]),np.array([x[:, 0], x[:, 1], x[:, 0]
** 2, x[:, 1] ** 2, x[:, 0] * x[:, 1] , x[: , 0]**2 * x[: , 1] , x[: , 0] * x[: , 1]**2 ,x[:
, 0]**3 , x[: , 1]**3  ]).T,np.zeros([len(x), 10])))
        B2 = np.column_stack((np.zeros([len(x), 10]), np.ones([len(x[:, 0]),
1]),np.array([x[:, 0], x[:, 1], x[:, 0] ** 2, x[:, 1] ** 2, x[:, 0] * x[:, 1] , x[: , 0]**2
* x[: , 1] , x[: , 0] * x[: , 1]**2 ,x[: , 0]**3 , x[: , 1]**3]).T))
        A = np.vstack((B1, B2))
        Params = np.linalg.inv(A.T @ A) @ (A.T @ y_label)

        xnew_train = Params[0] + Params[1] * x[:, 0] + Params[2] * x[:, 1] + Params[3] * x[:,
0] ** 2 + Params[4] * x[:,1] ** 2 + Params[5] * x[:, 0] * x[:, 1] + Params[6] * x[: , 0]**2
```

```
* x[: , 1] + Params[7] * x[: , 0] * x[: , 1]**2 + Params[8] * x[: , 0]**3 + Params[9] * x[:
, 1]**3
       ynew_train = Params[10] + Params[11] * x[:, 0] + Params[12] * x[:, 1] + Params[13] *
x[:, 0] ** 2 + Params[14] * x[:,1] ** 2 + Params[15] * x[:, 0] * x[:, 1] + Params[16] * x[:
, 0]**2 * x[: , 1] + Params[17] * x[: , 0] * x[: , 1]**2 + Params[18] * x[: , 0]**3 +
Params[19] * x[: , 1]**3

       xnew_test = Params[0] + Params[1] * X[:, 0] + Params[2] * X[:, 1] + Params[3] * X[:,
0] ** 2 + Params[4] * X[:,1] ** 2 + Params[5] * X[:, 0] * X[:, 1] + Params[6] * X[: , 0]**2
* X[: , 1] + Params[7] * X[: , 0] * X[: , 1]**2 + Params[8] * X[: , 0]**3 + Params[9] * X[:
, 1]**3
       ynew_test = Params[10] + Params[11] * X[:, 0] + Params[12] * X[:, 1] + Params[13] *
X[:, 0] ** 2 + Params[14] * X[:,1] ** 2 + Params[15] * X[:, 0] * X[:, 1] + Params[16] * X[:
, 0]**2 * X[: , 1] + Params[17] * X[: , 0] * X[: , 1]**2 + Params[18] * X[: , 0]**3 +
Params[19] * X[: , 1]**3


       res_image_train = np.column_stack((xnew_train, ynew_train)) - y
       res_image_test = np.column_stack((xnew_test, ynew_test)) - Y

       print('********************** train_res **********************')
       print(res_image_train)
       print('********************** test_res **********************')
       print(res_image_test)
       print('********************** train_rmse **********************')
       rmse_train = rmse(res_image_train)
       print(rmse_train)
       print('********************** test_rmse **********************')
       rmse_test = rmse(res_image_test)
       print(rmse_test)

def coord_transform(coords , geopath) :
   img = rio.open(geopath)
   left = img.bounds[0]
   bottom = img.bounds[1]
   right = img.bounds[2]
   top = img.bounds[3]
   dx = (right - left) / img.shape[0]
   dy = (top - bottom) / img.shape[1]
   coords[: , 0] = left + coords[: , 0] * dx
   coords[: , 1] = top - coords[: , 1] * dy
   return(coords)
```