Lab5.py

```
"""
CIVE 6374 – Optical Imaging Metrology
Professor: Dr. Craig Glennie
Author: Joshua Genova
Lab # 5
Description: Resection
Deadline: April 19, 2023 10:00 AM
"""
import numpy as np
from numpy.linalg import inv
import math
from math import sin, cos
from statistics import mean
import preLab5 as prelab5
from statistics import mean
import matplotlib.pyplot as plt

np.set_printoptions(linewidth=400)

class Resection:
    def __init__(self, x, y, Xo, Yo, Zo, c, S, format_size, sigma_obs):
        self.x = x
        self.y = y
        self.Xo = Xo
        self.Yo = Yo
        self.Zo = Zo
        self.c = c
        self.S = S
        self.format_size = format_size
        self.sigma_obs = sigma_obs
        self.P = 1/((sigma_obs*10**3)**2)*np.identity(len(x*2))
        self.xp = -0.006
        self.yp = 0.006

    def find_approx(self):
        a, b, delta_x, delta_y = prelab5.similarity_transform(self.x, self.y, self.Xo, self.Yo)
        X_c = delta_x
        Y_c = delta_y
        Z_c = self.c*math.sqrt(a**2 + b**2) + mean(Zo)
        w = 0
        p = 0
        k = math.atan2(b, a)
        return X_c, Y_c, Z_c, w, p, k

    def find_uvw(self, Xi, Yi, Zi, X_c, Y_c, Z_c, M):
        u = M[0][0]*(Xi - X_c) + M[0][1]*(Yi - Y_c) + M[0][2]*(Zi - Z_c)
        v = M[1][0]*(Xi - X_c) + M[1][1]*(Yi - Y_c) + M[1][2]*(Zi - Z_c)
        w = M[2][0]*(Xi - X_c) + M[2][1]*(Yi - Y_c) + M[2][2]*(Zi - Z_c)
        return u, v, w

    def find_partials(self, Xi, Yi, Zi, X_c, Y_c, Z_c, w, p, k):
        M = prelab5.rot_mat(w, p, k)
```

```
        U, V, W = self.find_uvw(Xi, Yi, Zi, X_c, Y_c, Z_c, M)

        # dx
        dx_dXc = (-self.c/W**2) * (M[2][0]*U - M[0][0]*W)
        dx_dYc = (-self.c/W**2) * (M[2][1]*U - M[0][1]*W)
        dx_dZc = (-self.c/W**2) * (M[2][2]*U - M[0][2]*W)
        dx_dw = (-self.c/W**2)*((Yi - Y_c)*(U*M[2][2] - W*M[0][2]) - (Zi - Z_c)*(U*M[2][1] - 
W*M[0][1]))
        dx_dp = (-self.c/W**2)*((Xi - X_c)*(-W*sin(p)*cos(k) - U*cos(p)) + (Yi - Y_c)*
(W*sin(w)*cos(p)*cos(k) - U*sin(w)*sin(p)) + (Zi - Z_c)*(-W*cos(w)*cos(p)*cos(k) + 
U*cos(w)*sin(p)))
        dx_dk = -self.c*V/W
        dx = [dx_dXc, dx_dYc, dx_dZc, dx_dw, dx_dp, dx_dk]

        #dy
        dy_dXc = (-self.c/W**2) * (M[2][0]*V - M[1][0]*W)
        dy_dYc = (-self.c/W**2) * (M[2][1]*V - M[1][1]*W)
        dy_dZc = (-self.c/W**2) * (M[2][2]*V - M[1][2]*W)
        dy_dw = (-self.c/W**2)*((Yi - Y_c)*(V*M[2][2] - W*M[1][2]) - (Zi - Z_c)*(V*M[2][1] - 
W*M[1][1]))
        dy_dp = (-self.c/W**2)*((Xi - X_c)*(W*sin(p)*sin(k) - V*cos(p)) + (Yi - Y_c)*(-
W*sin(w)*cos(p)*sin(k) - V*sin(w)*sin(p)) + (Zi - Z_c)*(W*cos(w)*cos(p)*sin(k) + 
V*cos(w)*sin(p)))
        dy_dk = self.c*U/W
        dy = [dy_dXc, dy_dYc, dy_dZc, dy_dw, dy_dp, dy_dk]

        x_ij = self.xp - self.c*U/W
        y_ij = self.yp - self.c*V/W

        return dx, dy, x_ij, y_ij

    def find_delta(self, X_c, Y_c, Z_c, w, p, k):
        A_mat = np.zeros(shape=(len(self.Xo*2), 6))
        misclosure = np.zeros(len(self.Xo*2))
        idx = 0
        for i in range(len(self.Xo)):
            dx, dy, x_ij, y_ij = self.find_partials(self.Xo[i], self.Yo[i], self.Zo[i], X_c, Y_c, 
Z_c, w, p, k)
            A_mat[idx] = dx
            A_mat[idx+1] = dy

            misclosure[idx] = x_ij - self.x[i]
            misclosure[idx+1] = y_ij - self.y[i]
            idx += 2

        delta = -np.dot(np.dot(np.dot(inv(np.dot(np.dot(A_mat.T, self.P), A_mat)), A_mat.T), 
self.P), misclosure)
        X_c = X_c + delta[0]
        Y_c = Y_c + delta[1]
        Z_c = Z_c + delta[2]
        w = w + delta[3]
        p = p + delta[4]
        k = k + delta[5]

        return X_c, Y_c, Z_c, w, p, k, A_mat

    def converge(self):
        r_obs = self.sigma_obs
```

```
        c = self.c*10**-3
        tol_coords = self.S*r_obs/10
        tol_tilt = r_obs/(10*c)
        x_max = (self.format_size/2*math.sqrt(2))*10**-3
        tol_k = r_obs/(10*x_max)
        print(f'Tol_coords: {round(tol_coords, 6)}')
        print(f'tol_tilt: {round(math.degrees(tol_tilt), 6)}')
        print(f'Tol_k: {round(math.degrees(tol_k), 6)}\n')

        iters = 10
        self.X_c, self.Y_c, self.Z_c, self.w, self.p, self.k = self.find_approx()
        for i in range(iters):
            X_c_old = self.X_c
            Y_c_old = self.Y_c
            Z_c_old = self.Z_c
            w_old = self.w
            p_old = self.p
            k_old = self.k

            self.X_c, self.Y_c, self.Z_c, self.w, self.p, self.k, self.A_mat= self.find_delta(self.X_c, self.Y_c, self.Z_c, self.w, self.p, self.k)


            if abs(self.X_c-X_c_old)<tol_coords and abs(self.Y_c-Y_c_old)<tol_coords and abs(self.Z_c-Z_c_old)<tol_coords and abs(self.w-w_old)<tol_tilt and abs(self.p-p_old)<tol_tilt and abs(self.k-k_old)<tol_k:
                print(f'Converged at {i+1} iterations!')
                break

        print(f'X_c: {round(self.X_c, 4)}')
        print(f'Y_c: {round(self.Y_c, 4)}')
        print(f'Z_c: {round(self.Z_c, 4)}')
        print(f'w: {round(math.degrees(self.w), 4)}')
        print(f'phi: {round(math.degrees(self.p), 4)}')
        print(f'kappa: {round(math.degrees(self.k), 4)}\n')

    def find_est(self, Xi, Yi, Zi, M):
        U, V, W = self.find_uvw(Xi, Yi, Zi, self.X_c, self.Y_c, self.Z_c, M)
        x = self.xp - self.c*U/W
        y = self.yp - self.c*V/W
        return x, y

    def resid(self):
        M = prelab5.rot_mat(self.w, self.p, self.k)
        res_x = np.zeros(len(self.Xo))
        res_y = np.zeros(len(self.Yo))

        x_rms = 0
        y_rms = 0

        idx = 0
        for i in range(len(self.Xo)):
            x_est, y_est = self.find_est(self.Xo[i], self.Yo[i], self.Zo[i], M)
            res_x[idx] = x_est - self.x[i]
            res_y[idx] = y_est - self.y[i]
            x_rms = x_rms + res_x[i]**2
            y_rms = y_rms + res_y[i]**2
```

```
            idx += 1

        x_rms = math.sqrt((1/len(self.Xo))*x_rms)
        y_rms = math.sqrt((1/len(self.Yo))*y_rms)

        print(f'Vx: {np.around(res_x, 6)}')
        print(f'Vy: {np.around(res_y, 6)}')
        print(f'x_rms: {round(x_rms, 6)}')
        print(f'y_rms: {round(y_rms, 6)}\n')

    def find_corr_matrix(self):
        S_mat = np.zeros(shape=(6,6))
        self.std = []
        C = inv(np.dot(np.dot(self.A_mat.T, self.P), self.A_mat))
        for i in range(len(C)):
            S_mat[i][i] = math.sqrt(C[i][i])
            self.std.append(math.sqrt(C[i][i]))

        S_inv = inv(S_mat)
        self.R = np.dot(S_inv, np.dot(C, S_inv))
        print(f'Correlation Coefficient Matrix: \n{np.around(self.R, 10)}\n')
        print(f'Standard Deviation: \n{np.around(self.std, 10)}\n')

    def store_corr_mat(self):
        return [self.R[0][1],
                self.R[0][2],
                self.R[0][3],
                self.R[0][4],
                self.R[0][5],
                self.R[1][2],
                self.R[1][3],
                self.R[1][4],
                self.R[1][5],
                self.R[2][3],
                self.R[2][4],
                self.R[2][5],
                self.R[3][4],
                self.R[3][5],
                self.R[4][5]]

    def store_std_dev(self):
        return self.std

    def report(self):
        self.converge()
        self.resid()
        self.find_corr_matrix()

def plot_corr(corr_mat):
    x_vals = ["4", "5", "6", "7"]
    names = ["Xc_Yc", "Xc_Zc", "Xc_w", "Xc_p", "Xc_k", "Yc_Zc", "Yc_w", "Yc_p", "Yc_k", "Zc_w",
"Zc_p", "Zc_k", "w_p", "w_k", "p_k"]
    for i in range(len(names)):
        plt.plot(x_vals, corr_mat[i], '-o', label=names[i])

    plt.xlabel("# of Observed Image Points", fontdict=
{'family':'serif','color':'black','size':10})
```

```
    plt.ylabel('Correlation', fontdict={'family':'serif','color':'black','size':10})
    plt.title("Correlation Trend", fontdict ={'family':'serif','color':'black','size':12})
    plt.legend(loc='upper left')
    plt.show()

def plot_std(std_vec):
    x_vals = ["4", "5", "6", "7"]
    names = ["Xc", "Yc", "Zc", "w", "p", "k"]
    for i in range(len(names)):
        plt.plot(x_vals, std_vec[i], '-o', label=names[i])

    plt.xlabel("# of Observed Image Points", fontdict=
{'family':'serif','color':'black','size':10})
    plt.ylabel('Standard Deviation', fontdict={'family':'serif','color':'black','size':10})
    plt.title("Standard Deviation Trend", fontdict ={'family':'serif','color':'black','size':12})
    plt.legend(loc='upper left')
    plt.show()

if __name__=="__main__":
    # image 27
    x_27 = [-9.444, 18.919, 90.289, 18.174, 44.681, -7.578, 52.736]
    y_27 = [96.236, -81.819, -91.049, 109.538, 7.483, -49.077, -93.140]
    # image 28
    x_28 = [-105.378, -72.539, -1.405, -77.840, -48.786, -98.814, -38.924]
    y_28 = [98.756, -79.786, -86.941, 113.375, 10.165, -48.039, -90.035]
    # control points
    Xo = [-399.28, 475.55, 517.62, -466.39, 42.73, 321.09, 527.78]
    Yo = [-679.72, -538.18, -194.43, -542.31, -412.19, -667.45, -375.72]
    Zo = [1090.96, 1090.5, 1090.65, 1091.55, 1090.82, 1083.49, 1092]

    c = 153.358 # mm
    format_size = 228.6 # mm
    S = 5000
    sigma_obs = 6e-6
    corr_27 = []
    corr_28 = []

    std_27 = []
    std_28 = []
    for i in range(4):
        resection_27 = Resection(x_27[0:4+i], y_27[0:4+i], Xo[0:4+i], Yo[0:4+i], Zo[0:4+i], c, S,
format_size, sigma_obs)
        print('-'*80)
        print(f'Printing Report for Image 27, points 100, 104, 105, 200 - 20{i}\n')
        print('-'*80)
        resection_27.report()
        corr_27.append(resection_27.store_corr_mat())
        std_27.append(resection_27.store_std_dev())
        print('-'*80)

        resection_28 = Resection(x_28[0:4+i], y_28[0:4+i], Xo[0:4+i], Yo[0:4+i], Zo[0:4+i], c, S,
format_size, sigma_obs)
        print('-'*80)
        print(f'Printing Report for Image 28, points 100, 104, 105, 200 - 20{i}\n')
        print('-'*80)
        resection_28.report()
        corr_28.append(resection_28.store_corr_mat())
```

```
        std_28.append(resection_28.store_std_dev())
        print('-'*80)

    # plot correlation coefficient matrix trend
    corr_27 = np.array(corr_27).T
    plot_corr(corr_27)
    corr_28 = np.array(corr_28).T
    plot_corr(corr_28)

    # plot standard deviation trend
    std_27 = np.array(std_27).T
    plot_std(std_27)
    std_28 = np.array(std_28).T
    plot_std(std_28)
```