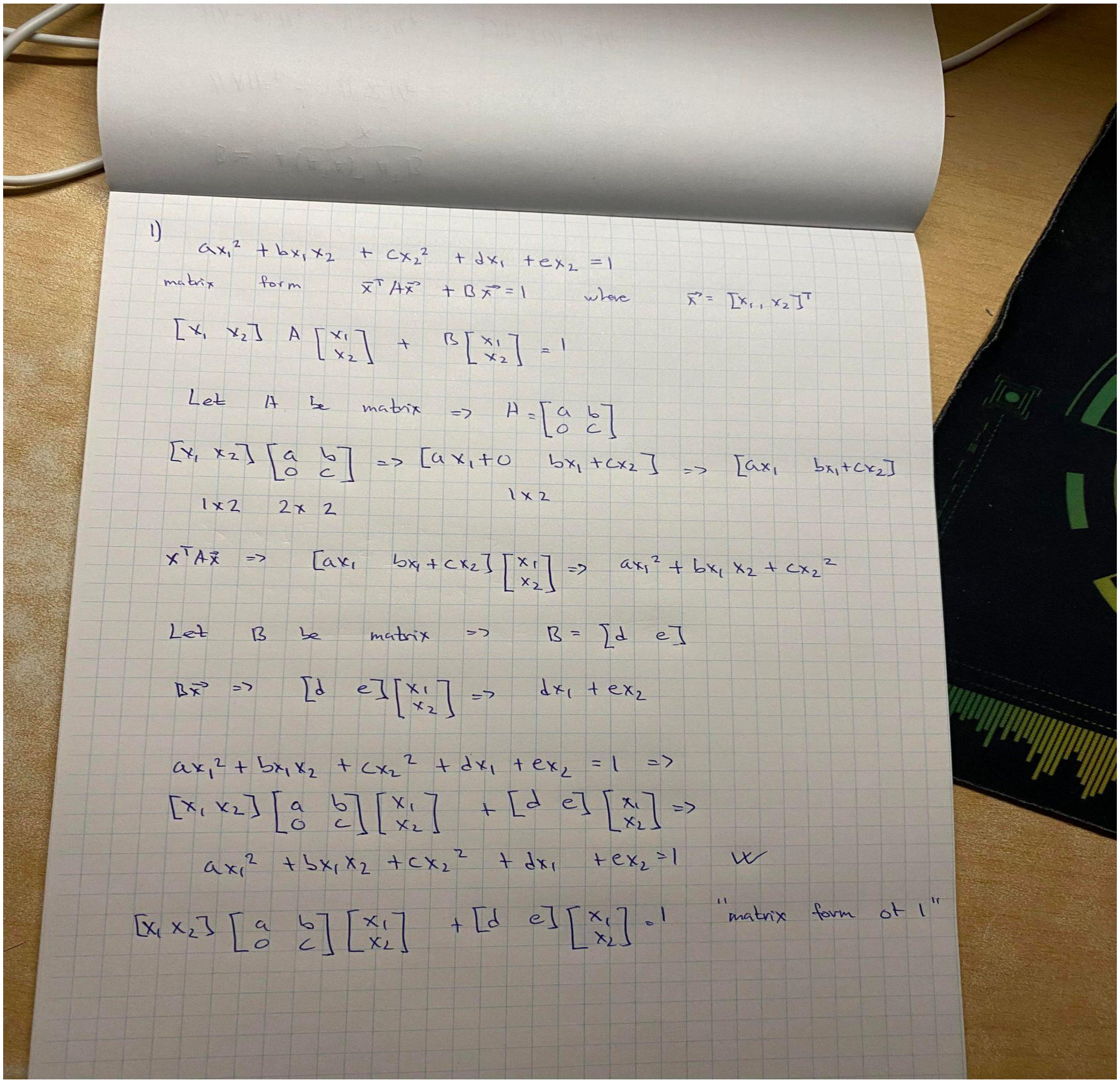
1) **Read textbook 6.6, and express (1) in the matrix form**



•What are matrix A and B?

A is an n × n symmetric matrix, B is a 1 × n matrix. Whereas they help us associate the n number of unknowns in a quadratic equation. Thus, on A matrix being singular or not, we can determine if exactly 1 eigenvalues has a value of 0 and possibly reduce the quadratic equation for simpler calculation.

• How to use A to determine which conic section the quadratic form represents?

To determine if a quadratic equation represents a circle, ellipse, hyperbola or a parabola using A, we could multiply by the value C from the equation

$ax_1^2 + bx_1 * x_2 + cx_2^2 + dx_1 + ex_2 = 1$ whereas C is c and A is a. In other words, if

$a = c$ is a circle

$a * c = 0$ is a parabola

$a * c \geq 0$ is an ellipse

$a * c < 0$ is hyperbola.

Thus, from the matrix A we can simply know our variables a and c.

2)Implement algorithm 1.

```
# this is for algorithm 1
def rotation(xp):
    # Compute the rotation for general_to_standard
    # The input model is
    #         xp[0]xx+xp[1]xy+xp[2]yy = 1
    # find the rotation matrix that make it
    #        xx/alpha^{2}  + yy/beta^{2} = 1
    #
    # returns
    # U : rotation matrix
    # P : [alpha, beta], where alpha and beta are the parameters
    #     for the standard model
    #
    # TODO: conplete this function and replace the return values
    A = np.array([[xp[0], xp[1]/2],
                  [xp[1]/2, xp[2]]])
    ((alpha, beta), U) = np.linalg.eig(A)

    #[[0.75306248 -0.65794901]
    #[ 0.65794901 0.75306248]]
    alpha = 1/math.sqrt(alpha)
    beta = 1/math.sqrt(beta)
    return (U,alpha,beta)
```

```python
    def translation(x) :
        # do the translation for general_to_standard
        # input x means
        #       x[0] xx + x[1] xy + x[2] yy + x[3]x + x[4] y = 1
        # for the general model
        # output xp means
        #      xp = [a, b, c, z, w]
        # for the semi-standard model
        # a(x-z)(x-z) + b(x-z)(y-w) + c(y-w)(y-w) = 1
        #
        # TODO: complete this part and replace the return values
        #
        #np.linalg.solve()
        a = x[0]
        b = x[1]
        c = x[2]
        d = x[3]
        e = x[4]
        A = np.array([[(-2)*a, (-1)*b],
                      [(-1)*b, (-2)*c]])
        B = np.array([d, e])
        xp = np.linalg.solve(A, B)
        #sol = np.array([a,b,c,xp[0],xp[1]])

        #New matrix
        # z=xp[0] , w =xp[1]
        Z = np.array([[(xp[0]*xp[0])+(1/a), (xp[0]*xp[1]), (xp[1]*xp[1])],
                       [(xp[0]*xp[0]),(xp[0]*xp[1])+(1/b), (xp[1]*xp[1])],
                       [(xp[0]*xp[0]),(xp[0]*xp[1]), (xp[1]*xp[1])+(1/c)]])
        result = np.array([1,1,1])
        y = np.linalg.solve(Z,result)
        #return np.array([0.00141895161, -0.00106665618, 0.00127449289, 46.2251570, 59.9276681])
        sol = np.array([y[0],y[1],y[2],xp[0],xp[1]])
        return sol
    # this is for algorithm 1
```

3) Implement algorithm 2.

```python
    return alpha, beta

# this is for algorithm 2
def standard_to_general(means, U, alpha, beta):
    # convert
    #          xx/alpha^2 + yy/beta^2 = 1
    # to       a xx + bxy + cyy + dx + ey = 1
    # using rotation matrix U and translation means
    # return:
    #      coef = [a, b, c, d, e]
    #
    # TODO: complete this code and replace the return values
    alpha, beta, U = bigToSmall(alpha, beta, U)
    A = np.array([
                  [1/(alpha*alpha), 0],
                  [0, 1/(beta*beta)]
    ])
    S = np.dot(np.dot(U.T,A),U)
    p = (1 - A[0,0]/(means[0]*means[0])) - ((A[0,1]*2)*means[0]*means[1]) - (A[1,1]*means[1]*means[1])
    a = (A[0,0])/p
    b = (A[0,1]*2)/p
    c = A[1,1]/p
    d = (-2*A[0,0]*means[0]-A[0,1]*means[1])/p
    e = (-A[0,1]*means[0] - 2*A[1,1]*means[1])/p
    #38.18272881937494 22.967166644060654 [[-0.73960154 -0.67304499]
    #[ 0.67304499 -0.73960154]]
    return np.array([a,b,c,d,e])
```

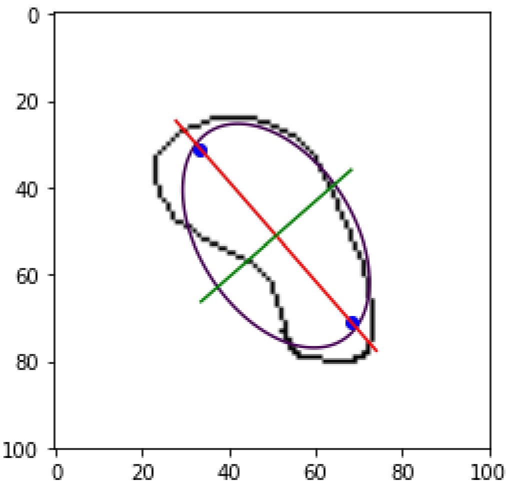
4)Give one or two examples to compare algorithm 1 and algorithm 2.

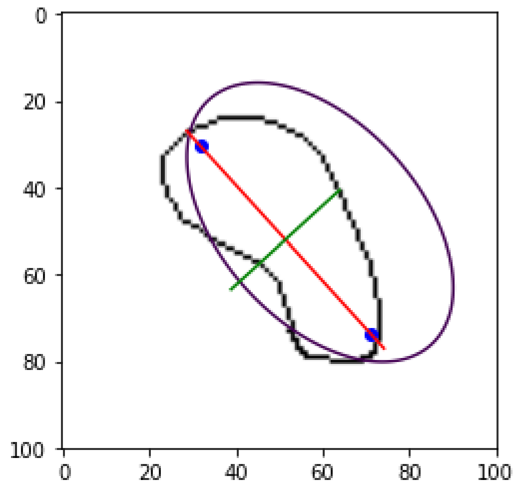
From the results, can you say which algorithm is more accurate?

From the following examples of conic section drawings,

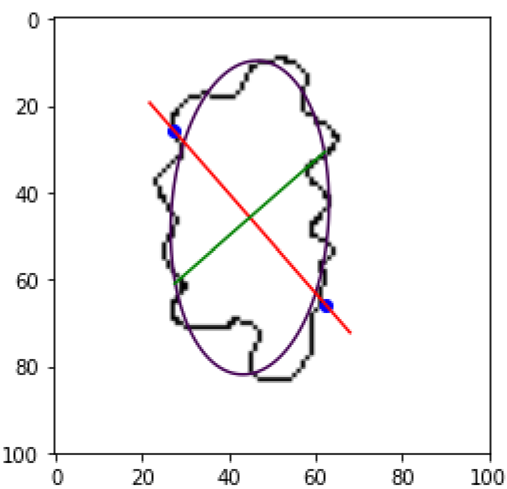
Algorithm 1 **Algorithm 2**

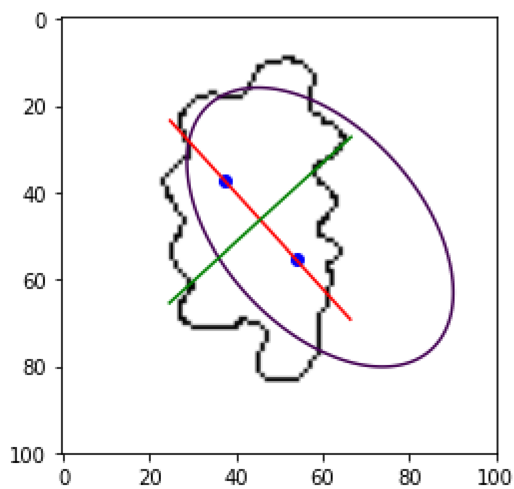
(1.1) (1.2)





(2.1) (2.2)





I would say Algorithm 2 is more accurate. Since it can be seen that it gives the more accurate drawings. Although Algorithm 2 calculates more accurately on picture (1.2), it doesn't calculate as accurately as Algorithm 1 for picture (2.2). On that note, I would prefer Algorithm 1 rather than Algorithm 2 because of the fact it can be used to calculate more diverse drawings.

5) In numpy.linalg.lstsq, what is the purpose of returning an SVD? Explain its relation with rank. If numpy.linalg.lstsq already returns rank, why does it still need to return SVD?

Rank of the matrix tells us the number of non-zero singular values of the current matrix that we are trying to calculate. Thus,since we know the singular values of $\sum$, we can calculate our eigenvalues much more easily. Hence, the purpose of returning an SVD of a matrix is that there is no other matrix that can give a better approximation in terms of A. Future on we can use the Frobenius norm to apply to our real world applications with data science. In numpy.linalg.lstsq already returns rank as it shows how many singular values there are which gives us the index that we limit our program to calculate and it still need to return SVD because it gives the the singular values of the current matrix which will be useful to calculate SVD of matrix.