An Exploration of Coordinate Metrology

Katie Cakounes and Aimee Ross

August 5, 2009

Abstract

Many manufacturing companies use coordinate measuring techniques to test their products and make sure that they satisfy requirements. Sensing machines are used to record points on the product, and then the points are tested to see how close they are to being a certain shape (ex: circle, sphere, square, cone, etc.). In this paper, we will look at least squares fitting of shapes and apply algorithms to test points on different shapes. We will compare different methods (including Householders and Givens transformations) and different coordinate systems (cartesian vs. polar).

1 Introduction

Linear least squares problems are used to solve an overdetermined system of linear equations. In the application of finding the best geometric fit of a shape to a set of points, the sum of the perpendicular offsets must be minimized.[3] Normal equations are used to solve simple least squares problems in the following form

$$A^T Ax = A^T b \tag{1}$$

For more complex problems, a QR factorization and Singular Value Decomposition (SVD) can be used. There are various methods to obtain a QR factorization, each with its own advantages and disadvantages. We take this into account when writing the MATLAB code for least squares fit of different shapes.

2 QR Factorization

2.1 Householder Transformation

A QR factorization of a matrix A can be found using Householder transformations. An $n \times n$ matrix H of the form $H = I - \frac{2}{v^T v} vv^T$ is a Householder transformation. To show that a matrix is symmetric, we must prove that $H^T = H$. To show that a matrix is orthogonal, we must prove that $H^T H = I$. To prove the matrix H is symmetric and orthogonal:

$$\begin{aligned} H^T &= (I - 2uu^T)^T \\ &= I - 2(u^T)^T u^T \\ &= I - 2uu^T \\ &= H \end{aligned}$$

$$\begin{aligned} H^T H = H^2 &= (I - 2uu^T)(I - 2uu^T) \\ &= I - 2uu^T - 2uu^T + 4uu^T \\ &= I - 4uu^T + 4u(u^T u)u^T \\ &= I - 4uu^T + 4uIu^T \\ &= I - 4uu^T + 4uu^T \\ &= I \end{aligned}$$

Since H is both symmetric and orthogonal, then $H^T H = H^2$.

The following steps are used to obtain a Householder transformation: [2]

1. Given a vector $x \in \Re^n$, set $\alpha = ||x||_2$
2. Set $\beta = \alpha(\alpha - x_1)$
3. Set $v = (x_1 - \alpha, x_2, \ldots, x_n)$
4. Let $u = \frac{1}{||v||_2} v = \frac{1}{\sqrt{2\beta}} v$
5. Let $H = I - 2uu^T = I - \frac{1}{\beta} vv^T$

After obtaining a Householder transformation H_1, multiplying H_1A will show that all entries in the first column of the matrix were zeroed out except the first entry. The same process can be continued until an upper triangular matrix is obtained. This upper triangular matrix will be denoted as R. To find matrix Q, simply use matrix multiplication in the form $H_1H_2\cdots H_n = Q$. It can be verified that matrix $A = QR$.

2.2 Givens Transformations

Givens transformations can also be used to obtain a QR factorization of a matrix. Givens transformations can be implemented either as rotations or reflections. Givens transformations differ from Householder transformations in that they can zero out a single entry of a matrix. Let R denote the rotation matrix and G denote the reflection matrix:

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{2}$$

$$G = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ \sin(\theta) & -\cos(\theta) \end{bmatrix} \tag{3}$$

Both R and G are orthogonal matrices. To show a matrix is orthogonal, one needs only to show that $Q^TQ = I$. The following is a proof that matrix R is orthogonal:

$$R^TR = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

$$= \begin{bmatrix} \cos^2(\theta) + \sin^2(\theta) & -\cos(\theta)\sin(\theta) + \cos(\theta)\sin(\theta)) \\ \cos(\theta)\sin(\theta) - \cos(\theta)\sin(\theta)) & \cos^2(\theta) + \sin^2(\theta) \end{bmatrix}$$

From the trigonometric identity $\cos^2(\theta) + \sin^2(\theta) = 1$, we have $R^TR = I$. The same can be proven for G by a similar method. It can also be proven that matrix G is symmetic, i.e. $G = G^T$ For a rotation R, we set $\cos(\theta) = \frac{x_1}{r}$ and $\sin(\theta) = -\frac{x_2}{r}$. For a reflection G, we set $\cos(\theta) = \frac{x_1}{r}$ and $\sin(\theta) = \frac{x_2}{r}$.

Rotations and reflections can both be used to obtain a QR factorization of a matrix. If we index each transformation by the form G_{nm} where n represents the row and m represents the column of the entry that is zeroed

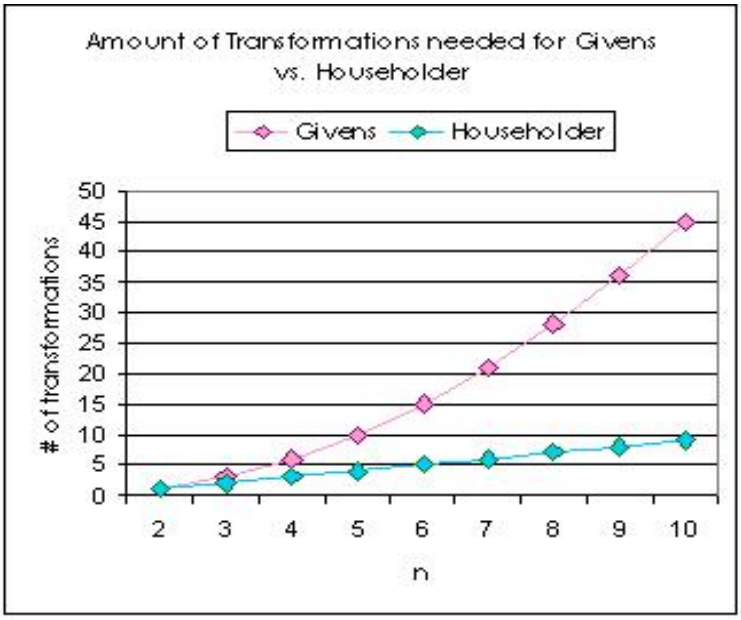

Figure 1: Householder vs. Givens

out at that step, we can obtain the following for a 3x3 matrix:

$$G_{31}G_{21}A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{23} \\ 0 & a_{32} & a_{33} \end{bmatrix}$$

At the next step we will zero out the last component using G_{32}:

$$G_{32}G_{31}G_{21}A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{bmatrix} = R$$

The R obtained is an upper triangular matrix used in the QR factorization. Since $A = QR$, we can let $Q^T = G_{32}G_{31}G_{21}$. We then have a system of the form $Rx = Q^T b$ and this system can be solved by back substitution. [2]

2.3 Comparison of Methods

MATLAB has a built in function to find the QR factorization of a matrix: $[Q, R] = qr(A)$. This method uses Householder transformations. In order

to decide which method is more efficient, it is important to look at the matrix that will be factored and the problem that is presented. If using these transformations for a different purpose than completing a QR factorization, Givens transformations could be more useful than Householder transformations because they can zero out a single entry of a matrix. If using these transformations for QR factorizations only, Householder transformations are more efficient because they require less steps and result in less error. Figure 1 illustrates this.

3 Fitting Shapes

We wrote codes for creating a least squares fit for different shapes to sets of data. Each code uses a similar method involving a QR factorization and singular value decomposition.

3.1 Circle

The equation for a circle is

$$(x - c_1)^2 + (y - c_2)^2 = r^2 \tag{4}$$

where (c_1, c_2) is the center and r is the radius. To fit a circle to n data points, the equations must be rewritten so that a center and radius can be found:

$$\begin{aligned} (x - c_1)^2 + (y - c_2)^2 &= r^2 \\ x^2 - 2c_1x + c_1^2 + y^2 - 2c_2y + c_2^2 &= r^2 \\ x^2 + y^2 &= r^2 - c_1^2 - c_2^2 + 2c_1x + 2c_2y \\ x^2 + y^2 &= c_3 + 2c_1x + 2c_2y \qquad c_3 = r^2 - c_1^2 - c_2^2 \end{aligned}$$

When the data points are substituted in for x and y, an overdetermined system is found: [2]

$$\begin{pmatrix} 2x_1 & 2y_1 & 1 \\ \vdots & \vdots & \vdots \\ 2x_n & 2y_n & 1 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} x_1^2 + y_1^2 \\ \vdots \\ x_n^2 + y_n^2 \end{pmatrix} \qquad C = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} \tag{5}$$

After C is solved, the radius can be determined from c_3 by substituting it into the formula for r^2. Figure 2 shows the least squares fit for a circle using the circfit.m file below.

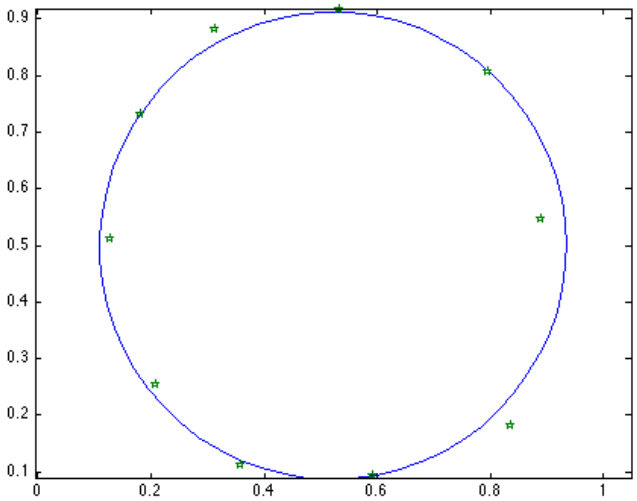

Figure 2: Least Squares Circle Fit

3.1.1 MATLAB Code

```
circfit.m
% This is our code for fitting the least squares circle
%    to a set of data points:

% Input the x and y coordinates of points you want to fit
%    into vectors.
% Type a 0 as a third input if you want to plot it.

function [c,r] = circfit(x,y,z)

n=size(x);

for i=1:n
    A(i,1)= 2*x(i);
    A(i,2)=2*y(i);
    A(i,3)=1;
    b(i,1) = (x(i))^2 + (y(i))^2;
end
```

```
C = A\b;

c = [C(1),C(2)];

r = sqrt(C(3)+(C(1))^2 + (C(2))^2)

if nargin == 3
    %plot the least squares circle and the original points
    t1 = 0:0.1:6.3;
    x1=c(1) + r*cos(t1);
    y1=c(2) + r*sin(t1);
    plot(x1,y1,x,y,p), axis equal
end

% Returns the radius of the circle, r
%   and the x and y coordinates of the center.
```

3.2 Concentric Circles

Two concentric circles can be fit by using the same center for both circles, but a different radius. The overdetermined system

$$\begin{pmatrix} 2x_1 & 2y_1 & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 2x_k & 2y_k & 1 & 0 \\ 2x_{k+1} & 2y_{k+1} & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ 2x_n & 2y_n & 0 & 1 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix} = \begin{pmatrix} x_1^2 + y_1^2 \\ \vdots \\ x_n^2 + y_n^2 \end{pmatrix} \qquad C = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix} \tag{6}$$

is solved to find C. (c_1, c_2) is the center of both circles. c_3 is used to find the radius of the circle fit to the first set of data, and c_4 is used to find the radius of the second circle. Figure 3 shows the image for least squares fit for concentric circles using the circfitcc.m file below.

3.2.1 MATLAB Code

`circfitcc.m`

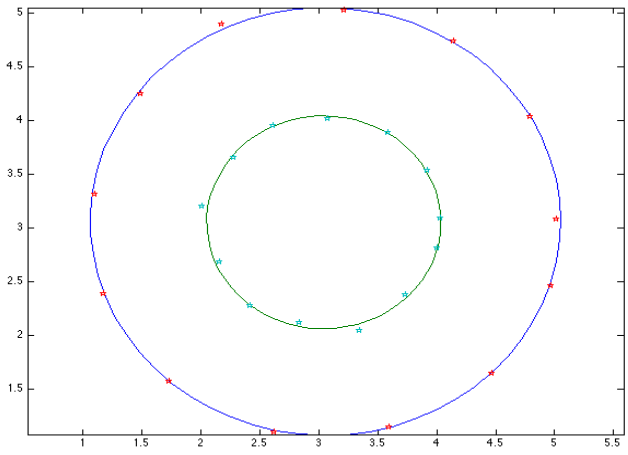

Figure 3: Least Squares Concentric Circle Fit

```
% Code for the least squres fit of two concentric circles,
%    using two sets of points. (one for each circle)

% Input the x and y coordinates of points you want to fit
%    into vectors.
% Type a 0 as a fifth input if you want to plot it.


function [c,r] = circfitcc(x,y,x1,y1,z)

n=size(x);

for i=1:n
    A(i,1)= 2*x(i);
    A(i,2)=2*y(i);
    A(i,3)=1;
    A(i,4)=0;
    b(i,1) = (x(i))^2 + (y(i))^2;
end
```

```
N = size(x1);

for i=n+1:n+N
    k = i-n(1);
    A(i,1)= 2*x1(k);
    A(i,2)=2*y1(k);
    A(i,3)=0;
    A(i,4)=1;
    b(i,1) = (x1(k))^2 + (y1(k))^2;
end

C = A\b;

c = [C(1),C(2)];

r1 = sqrt(C(3)+(C(1))^2 + (C(2))^2)
r2 = sqrt(C(4)+(C(1))^2 + (C(2))^2)

if nargin == 5
    %plot the least squares concentric circles and the original points
    t1 = 0:0.1:6.3;
    x2=c(1) + r1*cos(t1);
    y2=c(2) + r1*sin(t1);
    x3=c(1) + r2*cos(t1);
    y3=c(2) + r2*sin(t1);
    plot(x3,y3,x2,y2,x1,y1,p,x,y,p), axis equal

end

%Returns the radius of the circles, r1 and r2
% and the x and y coordinates of the center.
```

3.3 Line

Guided by a paper about some least squares problems by W. Gander and U. von Matt, we discovered a technique for fitting lines. A line is plotted using vectors of the form $c + n_1x + n_2y = 0$, where $n_1^2 + n_2^2 = 1$. [1] When

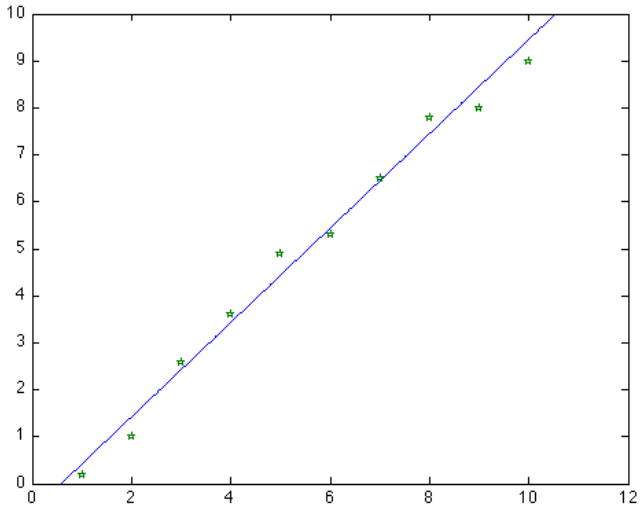

Figure 4: Least Squares Line Fit

finding a least squares fit to points, the residual distance from the point to the line is represented as $r = c + n_1 x_P + n_2 y_P$. The residual distance, r, must be minimized. So, substituting the data points into the equations, an overdetermined system is formed:

$$\begin{pmatrix} 1 & x_{p_1} & y_{p_1} \\ 1 & x_{p_2} & y_{p_2} \\ \vdots & \vdots & \vdots \\ 1 & x_{p_m} & y_{p_m} \end{pmatrix} \begin{pmatrix} c \\ n_1 \\ n_2 \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_m \end{pmatrix} \tag{7}$$

The linefit.m file below finds the best least squares fit line by solving the overdetermined system above using QR factorization and Singular Value Decomposition. Figure 4 shows the output of the program.

3.3.1 MATLAB Code

```
linefit.m
% To fit a line to a set of points

% Input the x and y coordinates of points you want to fit
```

```
%    into vectors.
% Type a 0 as a third input if you want to plot it.

function linefit(x,y,z)

m=size(x);

A = [ones(m,1) , x , y];

[Q,R] = qr(A);

[U,S,V] = svd(R(2:3,2:3));

n = V(:,2);

c = (-R(1,2)*n(1)  R(1,3)*n(2))/R(1,1);

if nargin == 3
    xmin = min(x);
    xmax = max(x);
    h = (xmax-xmin)/20;
    x1 = xmin:h:xmax;
    y1 = ((-n(1))/(n(2)))*x1  (c/(n(2)));
    plot(x1,y1,x,y,p)
end
```

3.4 Parallel Lines

Parallel lines are fit using the same basic method for fitting one straight line. The equations for the lines are $c_1 + n_1 x + n_2 y = 0$, $c_2 + n_1 x + n_2 y = 0$, with

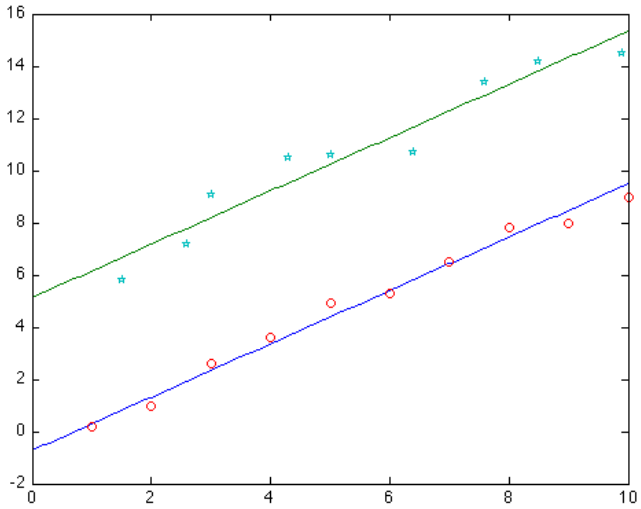

Figure 5: Least Squares Parallel Line Fit

$n_1^2 + n_2^2 = 1$. The overdetermined system we used is:

$$\begin{pmatrix} 1 & 0 & x_{p_1} & y_{p_1} \\ 1 & 0 & x_{p_2} & y_{p_2} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & x_{p_m} & y_{p_m} \\ 0 & 1 & x_{Q_1} & y_{Q_1} \\ 0 & 1 & x_{Q_2} & y_{Q_2} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & x_{Q_q} & y_{Q_q} \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ n_1 \\ n_2 \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_{p+q} \end{pmatrix} \tag{8}$$

The MATLAB code below solves this system and outputs the graph as seen in Figure 5.

3.4.1 MATLAB Code

```
% To fit two parallel lines to two sets of points.

% Input the x and y coordinates of points you want to fit
%    into vectors.
```

```
% Type a 0 as a fifth input if you want to plot it.

function [c,n] = plinefit(x,y,x1,y1,z)

m=size(x);
l=size(x1);

A1 = [ones(m,1), zeros(m,1), x, y];
A2= [zeros(l,1),ones(l,1), x1, y1];
A = [A1;A2];

[Q,R] = qr(A);

[U,S,V] = svd(R(3:4,3:4));

n = V(:,2);

c(1) = (-R(1,3)*n(1) -R(1,4)*n(2)) /R(1,1);
c(2) =(-R(2,3)*n(1) -R(2,4)*n(2)) /R(2,2);

if nargin == 5
    X = [x; x1];
    Y = [y;y1];

    xmin = min(X);
    xmax = max(X);
    h = (xmax-xmin)/20;
    x2 = xmin:h:xmax;

    y2 = ((-n(1))/(n(2)))*x2 - (c(1)/(n(2)));
    y3 = ((-n(1))/(n(2)))*x2 - (c(2)/(n(2)));

    plot(x2,y2,x2,y3,x,y,'o',x1,y1,'p')

end
```

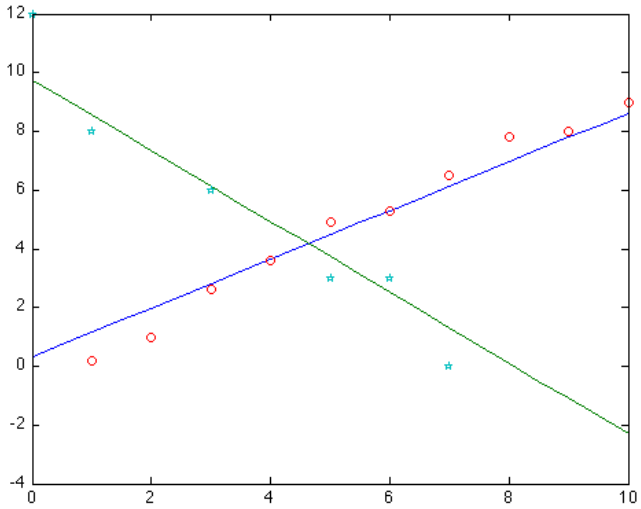

Figure 6: Least Squares Orthogonal Line Fit

3.5 Orthogonal Lines

Orthogonal lines are fit using almost the exact same equations for parallel lines. The difference between the two is that with parallel lines, the slopes of the lines are the same, but with orthogonal, the slopes of the lines are opposites. The equations for the two lines are $c_1 - n_2x + n_1y = 0$ and $c_2 - n_2x + n_1y = 0$, with $n_1^2 + n_2^2 = 1$. The overdetermined system used to find the least squares orthogonal lines is:

$$\begin{pmatrix} 1 & 0 & x_{p_1} & y_{p_1} \\ 1 & 0 & x_{p_2} & y_{p_2} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & x_{p_m} & y_{p_m} \\ 0 & 1 & y_{Q_1} & -x_{Q_1} \\ 0 & 1 & y_{Q_2} & -x_{Q_2} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & y_{Q_q} & -x_{Q_q} \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ n_1 \\ n_2 \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_{p+q} \end{pmatrix} \tag{9}$$

Figure 6 shows this fit using the olinefit.m file in the next section.

3.5.1 MATLAB Code

```
% To fit two orthogonal lines to two sets of points.

% Input the x and y coordinates of points you want to fit
%    into vectors.
% Type a 0 as a fifth input if you want to plot it.

function [c,n] = olinefit(x,y,x1,y1,z)

m=size(x);
l=size(x1);

A1 = [ones(m,1), zeros(m,1), x, y];
A2= [zeros(l,1),ones(l,1), y1, -x1];
A = [A1;A2];

[Q,R] = qr(A);

[U,S,V] = svd(R(3:4,3:4));

n = V(:,2);
n2(1) = -n(2);
n2(2) = n(1);

c(1) = (-R(1,3)*n(1) -R(1,4)*n(2)) /R(1,1);
c(2) =(-R(2,3)*n(1) -R(2,4)*n(2)) /R(2,2);

if nargin == 5

    X = [x; x1];
    Y = [y;y1];

    xmin = min(X);
    xmax = max(X);
    h = (xmax-xmin)/20;
    x2 = xmin:h:xmax;
```

```
    y2 = ((-n(1))/(n(2)))*x2 - (c(1)/(n(2)));
    y3 = ((-n2(1))/(n2(2)))*x2 - (c(2)/(n2(2)));

    plot(x2,y2,x2,y3,x,y,'o',x1,y1,'p')

    axis([min(X)-.1 max(X)+.1 min(Y)-.1 max(Y)+.1])
end
```

3.6 Rectangle

A rectangle is fit using this equation: [1]

$$\begin{pmatrix} 1 & 0 & 0 & 0 & x_{p_1} & y_{p_1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & 0 & 0 & x_{p_m} & y_{p_m} \\ 0 & 1 & 0 & 0 & y_{Q_1} & -x_{Q_1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & 0 & 0 & y_{Q_q} & -x_{Q_q} \\ 0 & 0 & 1 & 0 & x_{R_1} & y_{R_1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 1 & 0 & x_{R_r} & y_{R_r} \\ 0 & 0 & 0 & 1 & y_{S_1} & -x_{S_1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 1 & y_{S_s} & -x_{S_s} \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ n_1 \\ n_2 \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_{p+q+r+s} \end{pmatrix} \tag{10}$$

Our code has two graph output options. One graphs the four lines (each one either parallel or orthogonal to each other) separately, and the original points (shown in Figure 7). The other graph is of just the best-fit rectangle and the original points. We graphed this by finding the intersection of the lines, which are the corners of the rectangles. Then we graphed the four corner points only (shown in Figure 8).

3.6.1 MATLAB Code

`rectfit.m`

```
% This is our code for fitting the best geometric fit rectangle
```

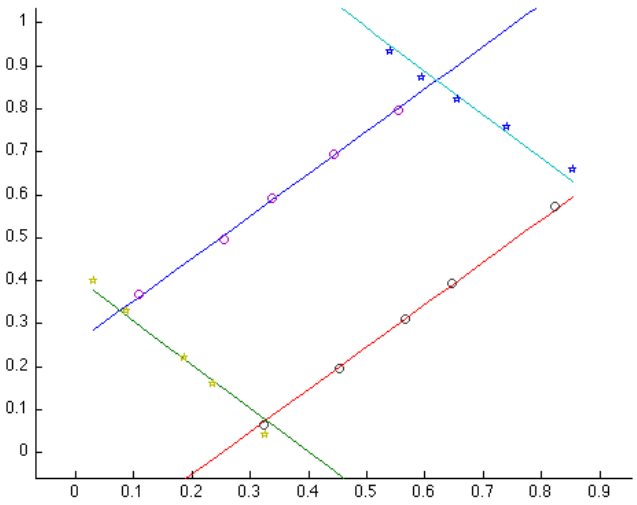

Figure 7: Least Squares Rectangle Fit

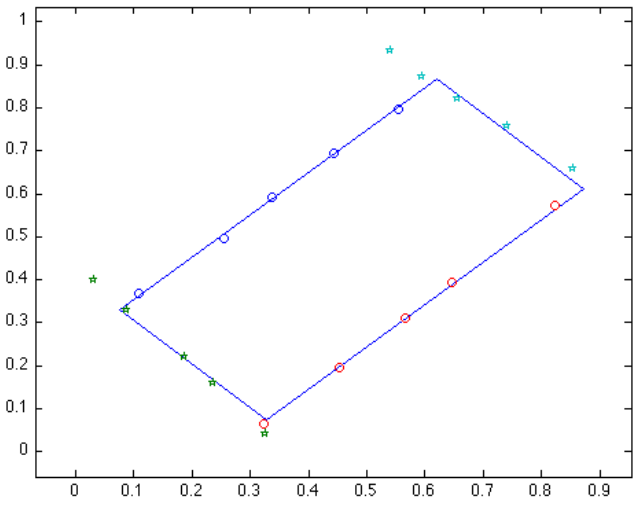

Figure 8: Least Squares Rectangle Fit

```
%     to a set of data points.  The points must be input in four
%     separate sets, one for each side of the rectangle. Each data
%     input must correspond to a side of the recange that is
%     orthogonal to the one before it

% Type any 9th input to plot the points and rectangle

function rectfit(x,y,x1,y1, x2, y2, x3, y3, z)

m=size(x);
l=size(x1);
k=size(x2);
j=size(x3);

A1 = [ones(m,1), zeros(m,1), zeros(m,1), zeros(m,1), x, y];
A2= [zeros(l,1),ones(l,1), zeros(l,1), zeros(l,1), y1, -x1];
A3 = [zeros(k,1), zeros(k,1), ones(k,1), zeros(k,1), x2, y2];
A4= [zeros(j,1), zeros(j,1), zeros(j,1),ones(j,1), y3, -x3];
A = [A1;A2;A3;A4];

[Q,R] = qr(A);

[U,S,V] = svd(R(5:6,5:6));

n = V(:,2);
n2(1) = -n(2);
n2(2) = n(1);

c(1) = (-R(1,5)*n(1) -R(1,6)*n(2)) /R(1,1);
c(2) =(-R(2,5)*n(1) -R(2,6)*n(2)) /R(2,2);
c(3) = (-R(3,5)*n(1) -R(3,6)*n(2)) /R(3,3);
c(4) =(-R(4,5)*n(1) -R(4,6)*n(2)) /R(4,4);

if nargin == 9
    X = [x; x1; x2; x3];
    Y = [y;y1;y2;y3];
    xmin = min(X);
    xmax = max(X);
```

```
    h = (xmax-xmin)/20;
    x4 = xmin:h:xmax;

    y4 = ((-n(1))/(n(2)))*x4  (c(1)/(n(2)));
    y5 = ((-n2(1))/(n2(2)))*x4  (c(2)/(n2(2)));
    y6 = ((-n(1))/(n(2)))*x4  (c(3)/(n(2)));
    y7 = ((-n2(1))/(n2(2)))*x4  (c(4)/(n2(2)));

 %  Use this plot only to show the parallel and orthogonal
 %      lines which best fit through the points
 %  plot(x4,y4,x4,y5,x4,y6,x4,y7)

    c=c;
    C = [n';n2];
    B = C;
    Z = -B*[c([1 3 3 1]); c([2 2 4 4])];
    Z=[Z Z(:,1)];

 %  Use this plot only to show the best fit rectangle
    plot(Z(1,:), Z(2,:))

    hold on
    plot(x,y,o,x1,y1,p,x2,y2,o,x3,y3,p)
    axis([min(X)-.1 max(X)+.1 min(Y)-.1 max(Y)+.1])

end
```

3.7 Square

Since a square is just a special type of rectangle, the codes for finding a least squares fit are very similar. We just had to make a constraint saying that each side of the rectangle must be the same (square). Figure 9 shows the fit.

4 Next Steps

Right now, we are working on creating least squares fits in three dimensions. Once we finish that, we will try to code our circle fit using polar coordinates.

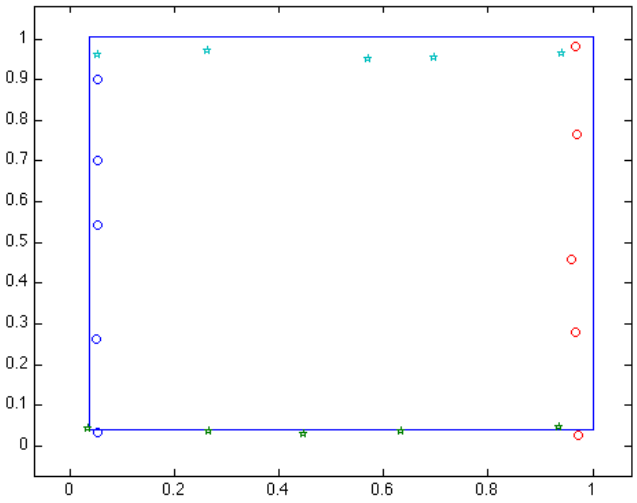

Figure 9: Least Squares Square Fit

4.1 Polar Coordinates

In two-dimensional rectangular cartesian coordinates, each point corresponds to a set of numerical coordinates that are the distance the point is from each axes (ex. (x, y)). The middle of the two axes is called the origin. In polar coordinates, each point is defined by a distance from a fixed point and angle from a fixed direction (ex. (r, θ)). The fixed point and direction are called the pole and polar axis, respectively.

It is fairly simple to convert between cartesian and polar coordinates.

$$x = r\cos\theta \qquad r = \sqrt{y^2 + x^2}$$
$$y = r\sin\theta \qquad \theta = \tan^{-1}(\frac{y}{x})$$

We would like to compare and contrast least squares fits using both types of coordinates, and find out if one is more efficient than the other.

5 Conclusion

Coordinate metrology using least squares fitting is an important topic in computational linear algebra. Various shapes, ranging from the basic line, to

the more complicated rectangle, can be fit to given sets of data. MATLAB codes using the QR factorization and Singular Value Decomposition (SVD) are used. The codes we implemented in this paper demonstrate only a few of the possibilities in coordinate metrology. We look forward to researching this topic further in the future.

6 Acknowledgements

We would like to thank Professor Steven Leon who guided us throughout this entire project, from proposing this research topic to providing the background we needed in computational linear algebra.

References

[1] Walter Gander. *Solving Problems in Scientific Computing Using Maple and MATLAB*. Springer, 4th edition, 2004.

[2] Steven J. Leon. *Linear Algebra with Applications*. Prentice Hall, 7th edition, June 2005.

[3] Gene H. Golub Walter Gander and Rolf Strebel. Least-squares fitting of circles and ellipses. Technical report, Departement Informatik, ETH Zurich, June 1994.