Math 521: Homework 1

Due on Thursday, February 15, 2018

Kristin Holmbeck

Contents

Theory **3**
Change of Basis . . . 3

Computing **3**
Unit Circle Transformations . . . 3
Principal Angles . . . 4
Image Transformations . . . 4
Scaling . . . 4
Translation . . . 5
Rotation . . . 5

Code **6**
prinAngles.m . . . 6
scale.m . . . 7
rotate.m . . . 8
translateH.m . . . 8
translateV.m . . . 8

List of Figures

1 Transformation of Points on the Unit Circle . . . 4
2 MATLAB's coins.png . . . 5
3 Image scaling about a point . . . 6
4 Rotation about a point . . . 6

Theory

Change of Basis

Let the basis $\mathcal{B}_1$ be the standard basis in $\mathbb{R}^2$, i.e., $\mathcal{B}_1 = span\{\vec{e_1}, \vec{e_2}\}$, and the basis $\mathcal{B}_2$ be given by $\mathcal{B}_2 = span\left\{\begin{bmatrix}1\\1\end{bmatrix}, \begin{bmatrix}-1\\1\end{bmatrix}\right\}$. Given $u_{\mathcal{B}_1} = (1,1)^T$, find $u_{\mathcal{B}_2}$.

$$V = \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}, W = \begin{bmatrix}1 & -1\\1 & 1\end{bmatrix} \implies W^{-1} = \frac{1}{2}\begin{bmatrix}1 & 1\\-1 & 1\end{bmatrix}$$

$$u_{\mathcal{B}_2} = W^{-1}V u_{\mathcal{B}_1} = \begin{bmatrix}1\\0\end{bmatrix}$$

Computing

Unit Circle Transformations

Write a code to generate 1000 random numbers contained on the unit circle. Apply several random matrices to this data and describe your results in the terminology of bases and change of bases. How do your results differ if the multiplying matrix is constrained to be orthogonal?

The following matrices are the transformations used in Figure 1, and note that A_3 is the Q matrix for the QR decomposition of A_1.

$$A_1 = \begin{bmatrix}0.524 & -0.038\\-0.401 & -0.210\end{bmatrix}, \quad A_2 = \begin{bmatrix}2 & 0\\0 & 2\end{bmatrix}, \quad A_3 = \begin{bmatrix}-0.794 & 0.608\\0.608 & 0.794\end{bmatrix}$$

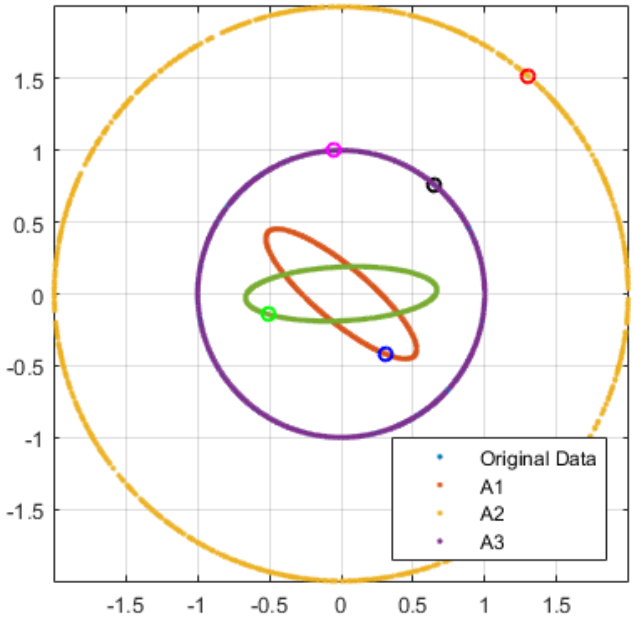

Figure 1: Transformation of Points on the Unit Circle

The actual data lies underneath the orthogonal, or A_3-transformed data on the graph. That is, the orthogonal transformation rotated the unit circle counter-clockwise. As expected, the scaling matrix A_2 uniformly scaled the points outwards (since the scaling factor was greater than 1).

The random matrix, A_1, scaled the unit circle non-uniformly into an ellipse. For extra fun, applying the orthogonal transformation to these elliptically-scaled points simply rotated the matrix.

Principal Angles

Image Transformations

Any linear transformation i $\mathbb{R}^2$ is represented with respect to homogeneous coordinates by a matrix of the form:

$$\begin{bmatrix} A_{2x2} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}$$

Image translation is simple, as every point is uniformly translated in either direction. For scaling and rotation, however, using a matrix of the form above is necessary.

For the following images, we used MATLAB's built-in image demo, coins.png (Figure 2).

Scaling

As shown in class, for scaling by s_x and s_y in the $x-$ and $y-$directions respectively, the corresponding scaling transformation matrix is given by

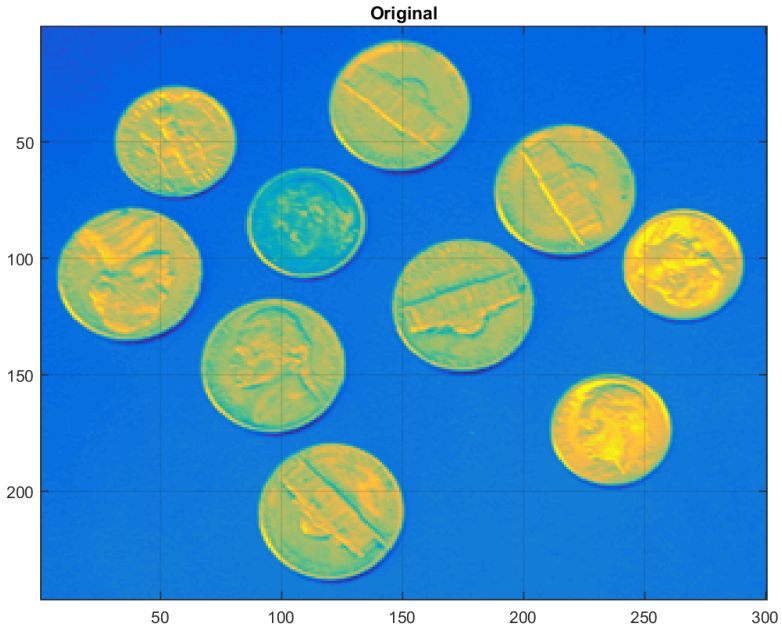

Figure 2: MATLAB's coins.png

$$M_{scale}(p) = \begin{bmatrix} s_x & 0 & (1-s_x)t_x \\ 0 & s_y & (1-s_y)t_y \\ 0 & 0 & 1 \end{bmatrix}$$

where $p = \begin{bmatrix} t_x & t_y & 1 \end{bmatrix}^T$. The data in Figure 3 was scaled by 2 in both x and y, about the point $(t_x, t_y) = (150, 50)$.

Translation

Rotation

As shown in class, for rotating by θ about a point $p = \begin{bmatrix} t_x & t_y & 1 \end{bmatrix}^T$, the rotation matrix is given by:

$$M_{rot}(p) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

So where do we account for p? The matrix above rotates points with respect to the origin. In this case, we can translate the image (preserving the entirety of the image, unlike the previous section) by $-p$, which will make the origin p. Rotate about the origin, then undo the $-p$ translation by p, yielding the desired image.

The data in Figure 3 was rotated by $\theta = \pi/2$, about the point $(t_x, t_y) = (150, 50)$.

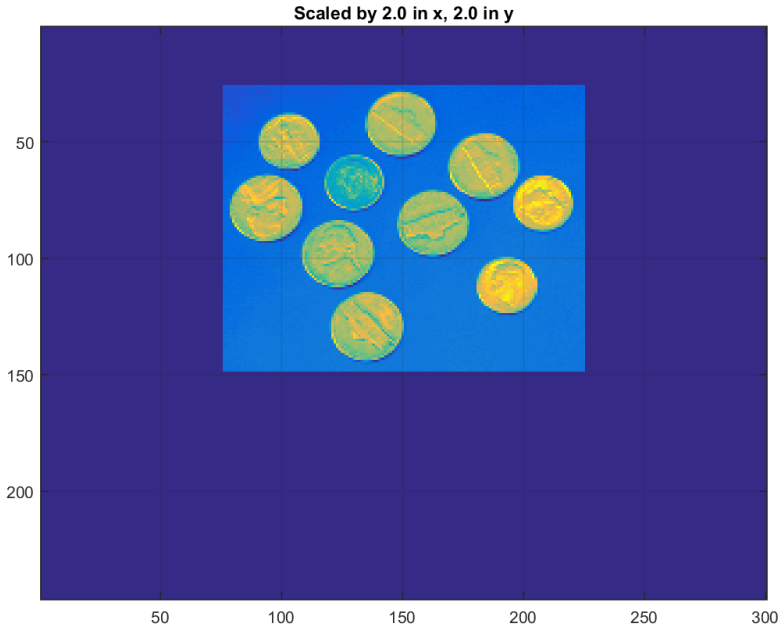

Figure 3: Image scaling about a point

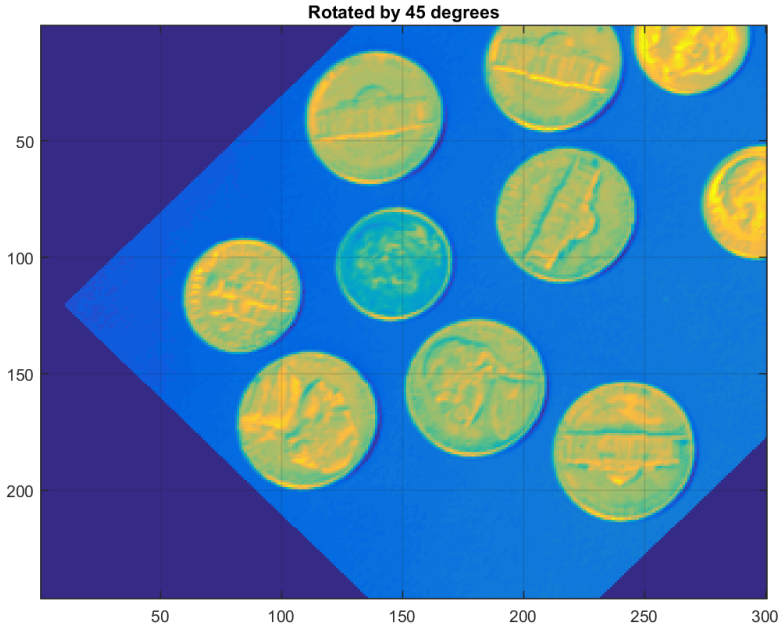

Figure 4: Rotation about a point

Code

prinAngles.m

```
function theta = prinAngles(X1,X2)

% Find orthonormal bases for X1,X2
[Q1,R1] = qr(X1,0);
[Q2,R2] = qr(X2,0);
% Q1 = orth(X1);
% Q2 = orth(X2);

% Compute SVD for cosine
[U,S,V] = svd(Q1' * Q2);
S = diag(S);

% Compute matrix Y
if rank(Q1) >= rank(Q2)
    Y = Q2 - Q1*(Q1'*Q2);
else
    Y = Q1 - Q2*(Q2'*Q1);
end

% SVD for sine
[U,M,V] = svd(Y,0);
M = diag(M);

% Compute the principal angles for k = 1,...,q
theta = zeros(size(X2,2), 1);
for ii = 1:length(theta)
    if S(ii)^2 < 1/2
        theta(ii) = acos(S(ii));
    end
    if M(ii)^2 <= 1/2
        theta(ii) = acos(M(ii));
    end
end

end
```

scale.m

```
function newImg = scale(img, alpha, P)
% SCALE
%
% Syntax:
%   newImg = SCALE(img, alpha, P)
%
% with
%   point to scale around: P = [tx; ty; 1]
%   scaling size: alpha = [sx; sy]
%

M = [alpha(1),  0,          (1-alpha(1))*P(1);
     0,         alpha(2),   (1-alpha(2))*P(2);
     0,         0,          1];

[nr,nc] = size(img);
[xs,ys] = meshgrid(1:nc, 1:nr);

xy_mat  = [xs(:)'; ys(:)'; ones(1,nr*nc)];

xyTrans = M*xy_mat;

xi = reshape(xyTrans(1,:), nr, nc);
yi = reshape(xyTrans(2,:), nr, nc);
```

```
25 newImg = interp2(xs,ys, img, xi, yi);
26
27 end
```

rotate.m

```
1  function newImg = rotate(img, theta, P)
2  % ROTATE
3  %
4  % Syntax:
5  %   newImg = ROTATE(img, theta, P)
6  %
7
8  M = [cos(theta),  -sin(theta), 0;
9       sin(theta),   cos(theta), 0;
10      0,             0,         1];
11
12 [nr,nc] = size(img);
13 [xs,ys] = meshgrid(1:nc, 1:nr);
14
15 % Subtract the xs and ys by the point P, i.e. center the point P as the
16 % origin
17 xy_mat  = [xs(:)'-P(1); ys(:)'-P(2); ones(1,nr*nc)];
18
19 xyTrans = M*xy_mat;
20
21 xi = reshape(xyTrans(1,:), nr, nc);
22 yi = reshape(xyTrans(2,:), nr, nc);
23
24 % Correct the point-P-at-the-origin while interp-ing
25 newImg = interp2(xs,ys, img, xi+P(1), yi+P(2));
26
27 end
```

translateH.m

```
1  function newImg = translateH(img, tx)
2
3  [nr, nc] = size(img);
4  newImg   = zeros(nr, nc);
5
6  if tx > 0
7      newImg(:,(tx+1):end) = img(:,1:(end-tx));
8  else
9      newImg(:,1:(end+tx)) = img(:,(abs(tx)+1):end);
10 end
11
12 end
```

translateV.m

```
1  function newImg = translateV(img, ty)
2
3  [nr, nc] = size(img);
4  newImg = zeros(nr, nc);
5
6  if ty > 0
7      newImg(ty+1:end,:) = img(1:end-ty,:);
8  else
9      newImg(1:(end+ty),:) = img(abs(ty)+1:end,:);
10 end
```

```
11
12 end
```