CMSC 828D: Fundamentals of Computer Vision
Homework 9

Instructors: Larry Davis, Ramani Duraiswami,
Daniel DeMenthon, and Yiannis Aloimonos
Based on homework submitted by Haiying Liu
Date: December 10, 2000

1. Indicate how you would fit a quadratic to a function …

Solution: Note that the quadratic function $p(x,y)=a_0+a_1x+a_2y+a_3xy+a_4x^2+a_5y^2$ has six unknowns. Given n points on this surface $(x_i,y_i,f_i), i=1,2,\cdots,n$, we can form a linear system: $\mathbf{Ma}=\mathbf{b}$, where

$$\mathbf{M}=\begin{bmatrix}1 & x_1 & y_1 & x_1y_1 & x_1^2 & y_1^2\\ 1 & x_2 & y_2 & x_2y_2 & x_2^2 & y_2^2\\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots\\ 1 & x_n & y_n & x_ny_n & x_n^2 & y_n^2\end{bmatrix},\ \mathbf{a}=\begin{bmatrix}a_0\\ a_1\\ \vdots\\ a_6\end{bmatrix},\ \mathbf{b}=\begin{bmatrix}f_1\\ \vdots\\ f_n\end{bmatrix}$$

When $n=6$ and $\text{rank}(\mathbf{M})=6$, the $\mathbf{a}$ can be uniquely solved by $\mathbf{a}=\mathbf{M}^{-1}\mathbf{b}$. Since the fitted surface $p(x,y)$ is a polynomial, it is easy to get its derivatives in the analytical form, e.g.

$$\frac{\partial p(x,y)}{\partial x}=a_1+a_3y+2a_4x,\ \frac{\partial p(x,y)}{\partial y}=a_2+a_3x+2a_5y,\ \ldots$$

When any two of the equations are dependent, $\mathbf{a}=\mathbf{M}^{-1}\mathbf{b}$ would fail, since $\mathbf{M}^{-1}$ does not exist. This could be tested by checking $\text{rank}(\mathbf{M})$. If $\text{rank}(\mathbf{M})<6$, the configuration is degenerate.

Some situations when this occurs are when there are less than 6 unique points, if all the points are collinear, 5 points are collinear, etc.

The Matlab script `hw9_1.m` is listed in appendix.

2. If you now had more than six points …

Solution: If we have more than six points, we can define an algebraic distance function as $d=\|\mathbf{Ma}-\mathbf{b}\|^2$, and obtain $\mathbf{a}$ by minimizing $d(\mathbf{a})$. This is a solution in the least squares sense.

If we decompose the $\mathbf{M}$ using the SVD, i.e. $\mathbf{M}=\mathbf{U}\,\mathbf{\Sigma}\,\mathbf{V}^{\mathrm{T}}$, we have regularized solution: $\mathbf{x}=\sum_{i=1}^{k}\frac{\mathbf{u}_i^{\mathrm{T}}}{\sigma_i}\mathbf{v}_i$, where $\sigma_1>\sigma_2>\cdots>\sigma_k>\varepsilon\geq\sigma_{k+1}\geq\sigma_{k+2}\geq\cdots\geq\sigma_n$, and $k=6$ in our case. The $\varepsilon>0$ is a small constant.

The Matlab script `hw9_2.m` is listed in appendix.

3. Generate several points along an ellipsoidal surface …

Solution: The Matlab script `hw9_3.m` is listed in appendix. All methods in `hw9_1.m`, `hw9_2.m`, `hw9_4.m` are tested. In `hw9_4.m`, the σ_i is randomly generated within range [0, 0.2]. The surfaces are plotted in [Figure 1].

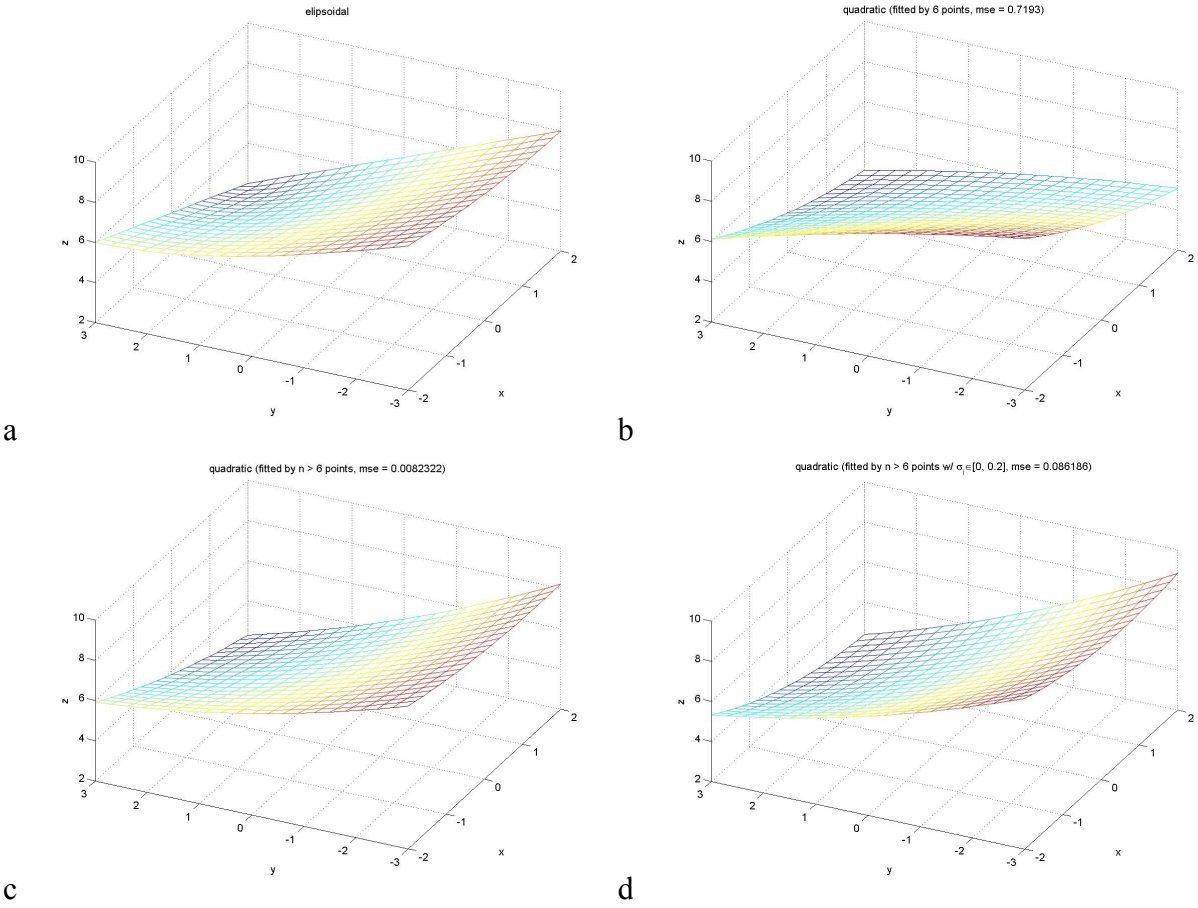


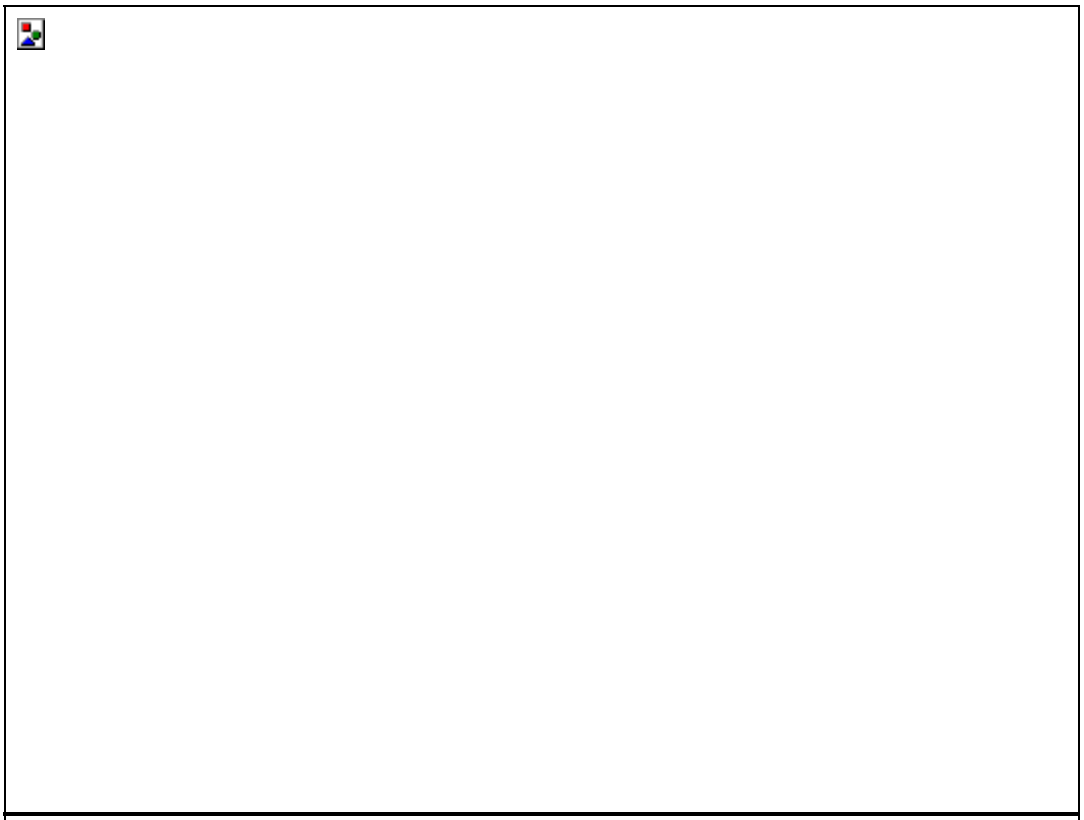
Figure 1: Ellipsoidal function and its fitted polynomial function. a) Ellipsoidal function. b) Fitted polynomial by 6 points. c) Fitted polynomial by all points at grid $[-2{:}0.2{:}2, -3{:}0.3{:}3]$. d) Fitted polynomial by all points at grid $[-2{:}0.2{:}2, -3{:}0.3{:}3]$ with standard deviations σ_i known.

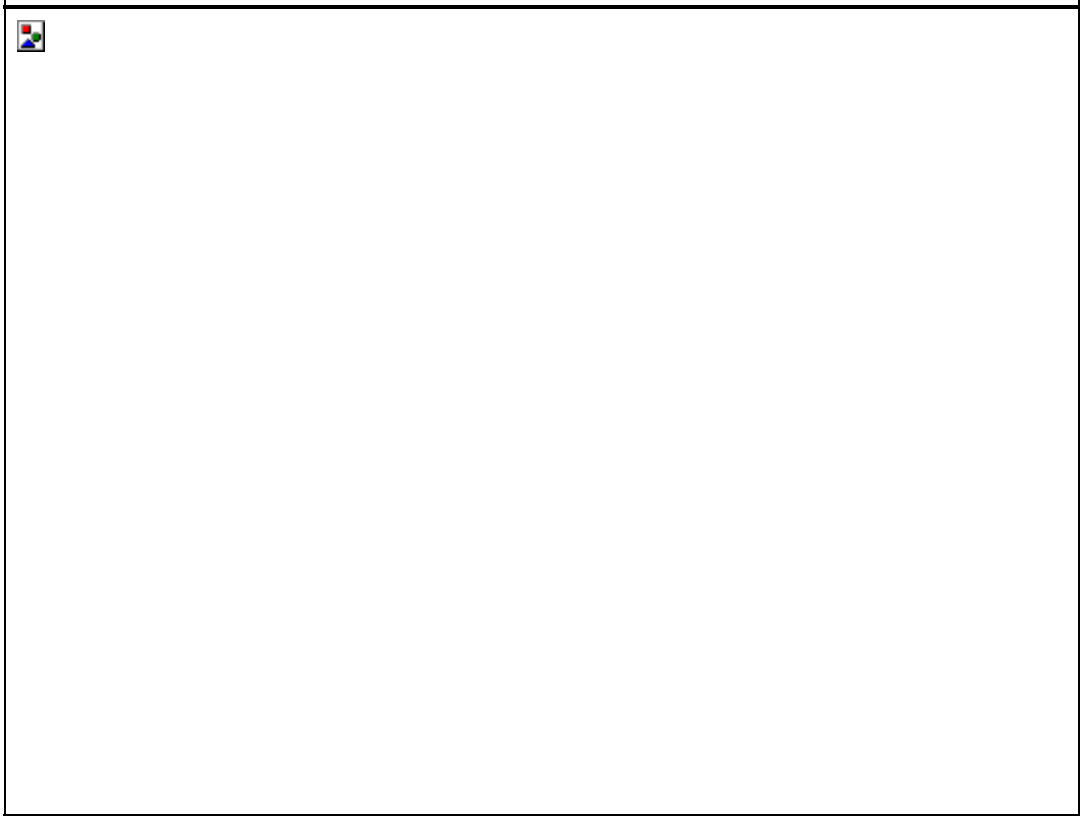
4. Now you are given standard deviations …

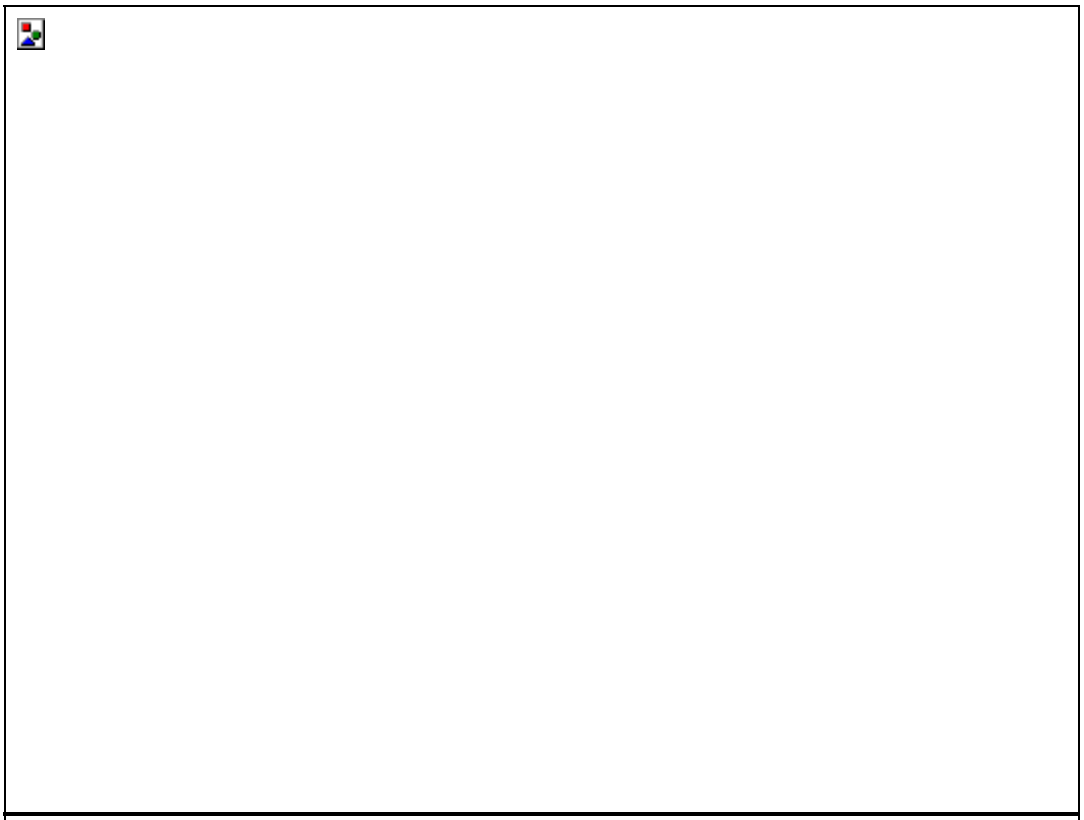
Solution: The Matlab script `hw9_4.m` is listed in appendix.

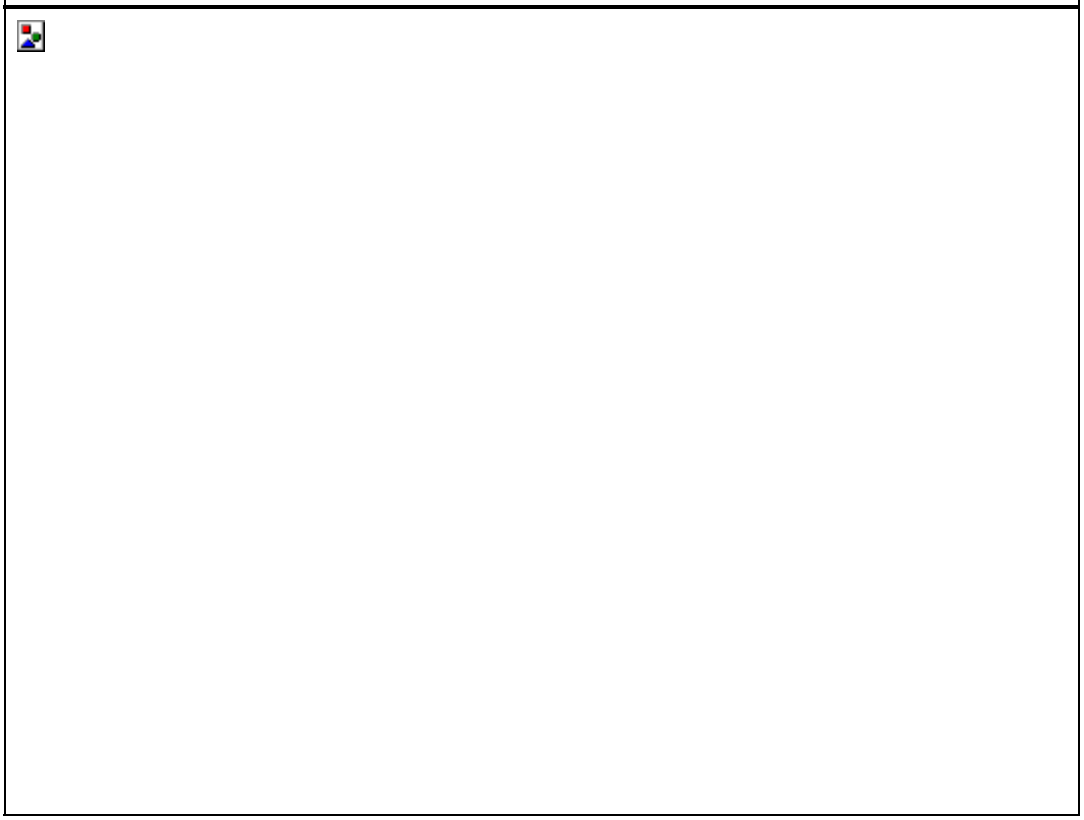
5. Write functions to compute the SSD score …

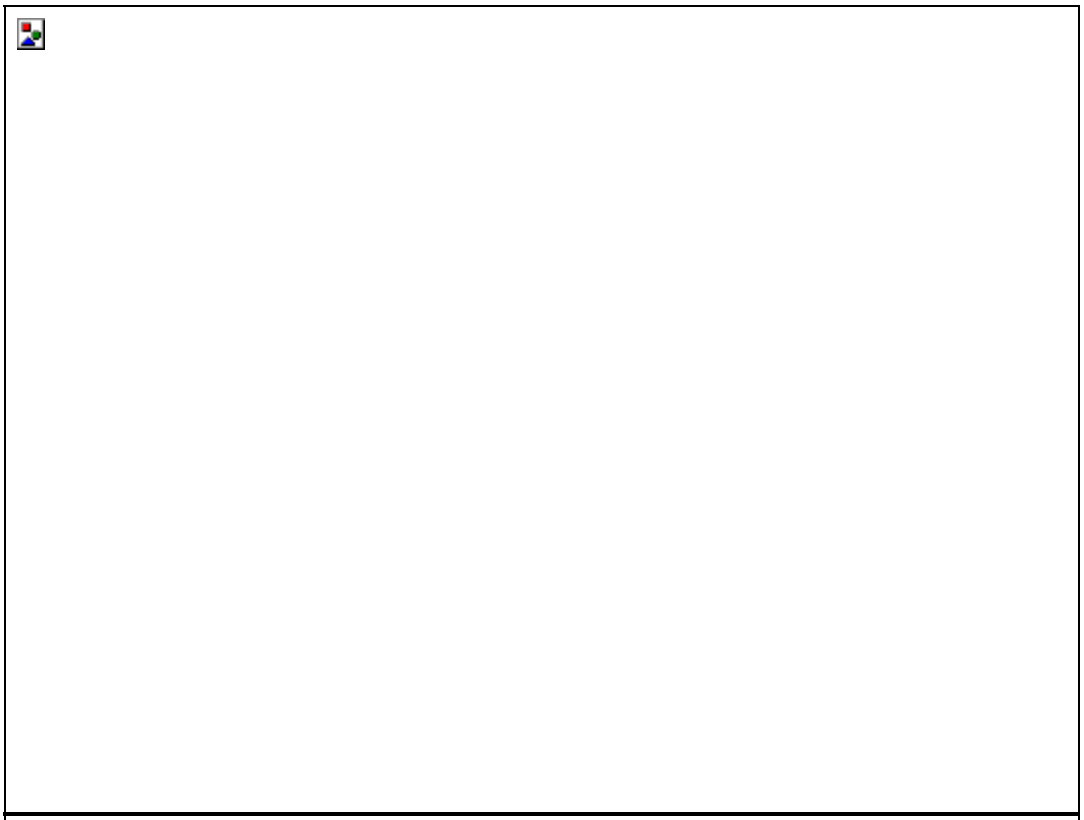
Solution: The Matlab script `hw9_5.m` is listed in appendix. The right image is used as reference image. Eight points are selected from row 15 to compute the SSD score and cross-correlation score. They are listed in [Figure 2].

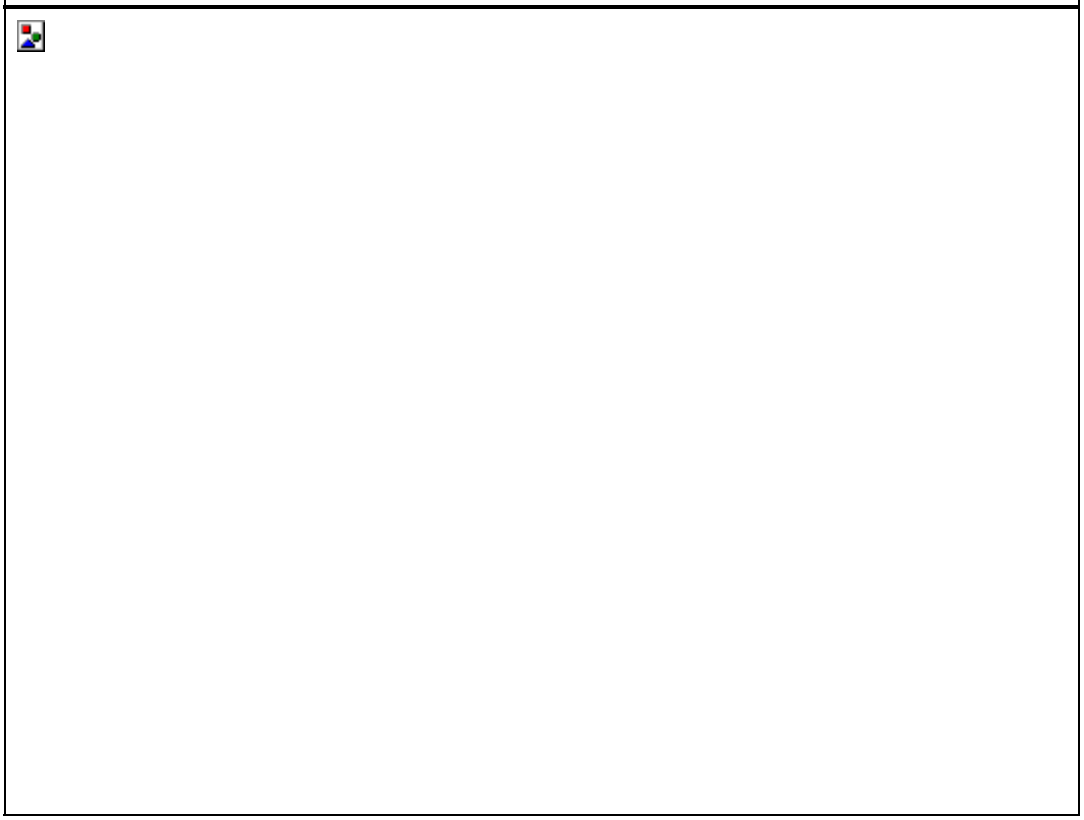












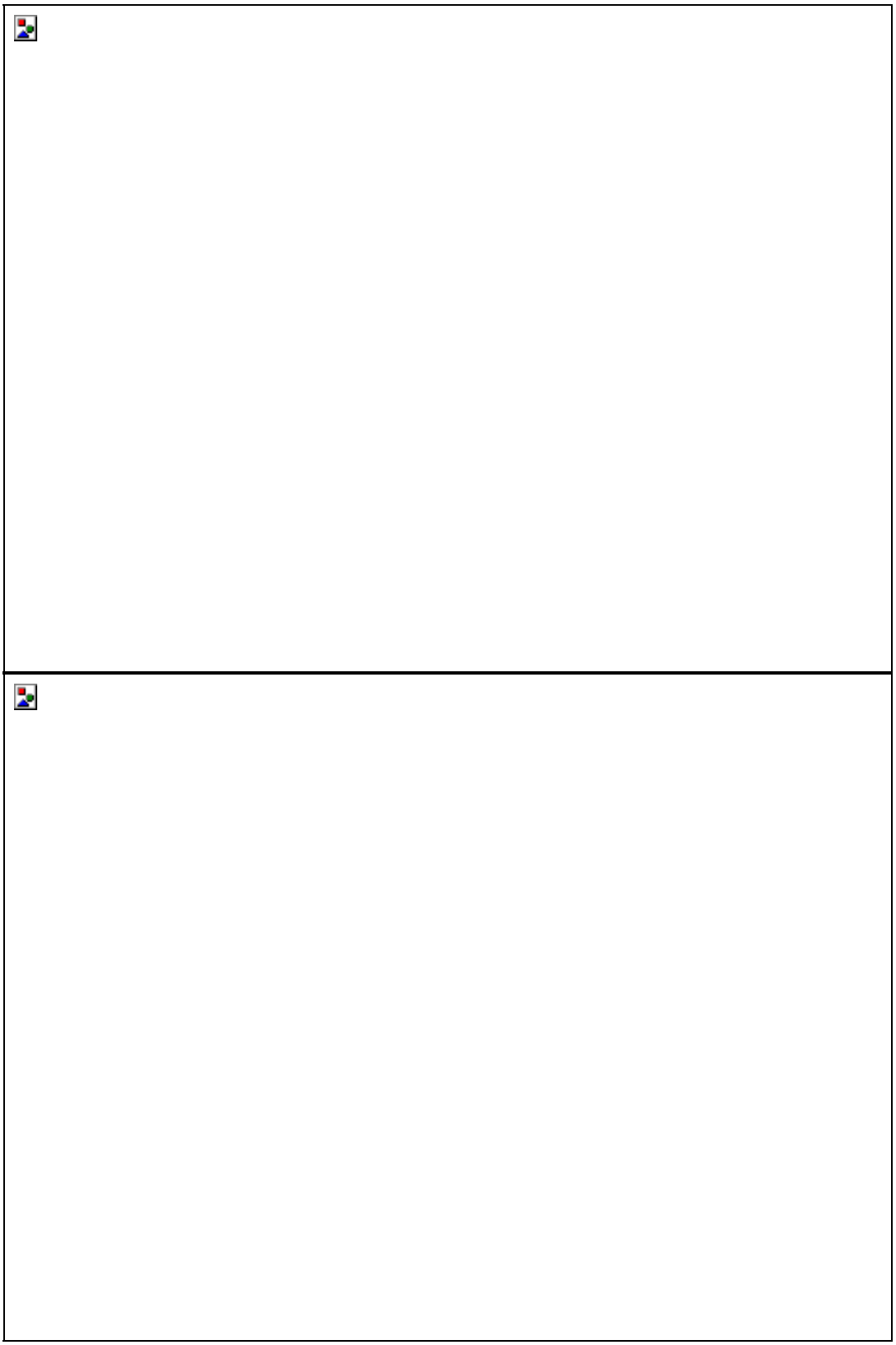


Figure 2: SSD score and cross-correlation score for 8 points on row 15.

Appendix:

- hw9_1.m

```
function coef = hw9_1(points)
% Syntax: coef = hw9_1(points)
%
%         points - 6 points known on the plane in the format
%                  [x1, y1, f1; x2, y2, f2; ...; x6, y6, f6]
%         coef   - coefficient (format [a0, ..., a5] for
%                  a0 + a1 * x + a2 * y + a3 * xy + a4x^2 + a5y^2
%
% Description: CMSC828D HW9_1
%
% Author: Haiying Liu
%   Date: Nov. 4, 2000
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

dbstop if error

msg = nargchk(1, 1, nargin);
if (~isempty(msg))
  error(strcat('ERROR:', msg));
end

clear msg;

%=========================================================================

% Compose the linear equations in the format Ax = b.
nPoints = size(points, 1);
if nPoints ~= 6
  error('ERROR: The number of points must be 6.');
end

A = zeros(nPoints, 6);
b = zeros(nPoints, 1);

for index = 1:nPoints
  x = points(index, 1);
  y = points(index, 2);
  A(index, :) = [1, x, y, x * y, x * x, y * y];
  f = points(index, 3);
  b(index)    = f;
end

% Determine if the 'A' is degenerate.
if rank(A) < min(nPoints, 6)
  error(['The points configuration is degenerate.\n', ...
      'There is no unique solution.']);
end

% Solve the equations if they are not degenerated.

coef = (A \ b)';
```

- hw9_2.m

```
function coef = hw9_2(points)
% Syntax: coef = hw9_2(points)
%
%         points - n points known on the plane in the format
%                  [x1, y1, f1; x2, y2, f2; ...; xn, yn, fn]
%         coef   - coefficient (format [a0, ..., a5] for
%                  a0 + a1 * x + a2 * y + a3 * xy + a4x^2 + a5y^2
%
% Description: CMSC828D HW9_2
%
% Author: Haiying Liu
%   Date: Nov. 4, 2000
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

dbstop if error

msg = nargchk(1, 1, nargin);
if (~isempty(msg))
  error(strcat('ERROR:', msg));
end

clear msg;

%=========================================================================

% Compose the linear equations in the format Ax = b.
nPoints = size(points, 1);
if nPoints < 6
  error('ERROR: The number of points must be larger than 6.');
end

A = zeros(nPoints, 6);
b = zeros(nPoints, 1);

for index = 1:nPoints
  x = points(index, 1);
  y = points(index, 2);
  A(index, :) = [1, x, y, x * y, x * x, y * y];
  f = points(index, 3);
  b(index)    = f;
end

% Solve the equations if they are not degenerated.
% distance = || Ax - b ||
% Minimize distance.
coef = solve_by_svd(A, b);
```

- hw9_3.m

```
function hw9_3
% Syntax: hw9_3
%
```

```
% Description: CMSC828D HW9_3
%
% Author: Haiying Liu
%   Date: Nov. 4, 2000
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

dbstop if error

msg = nargchk(0, 0, nargin);
if (~isempty(msg))
  error(strcat('ERROR:', msg));
end

clear msg;

%=========================================================================
%= Generate several points.

% Define view angle
global AZ;
global EL;
AZ = -64;
EL = 44;

% Reset random generator.
randn('state',sum(100 * clock));

% Generate points.
x             = -2:0.2:2;
y             = -3:0.3:3;
[X, Y]        = meshgrid(x, y);
inputs        = [X, Y];
Z_ellipsoidal = ellipsoidal(X, Y);

variance      = 0.2;
noise         = sqrt(variance) * randn(size(X));

Z_noise       = Z_ellipsoidal + noise;

points = [X(:), Y(:), Z_noise(:)]; % [x, y, z]

% Draw elipsoidal surface
figure;
mesh(X, Y, Z_ellipsoidal);
xlabel('x');
ylabel('y');
zlabel('z');
title('elipsoidal');
view(AZ, EL);
print -djpeg hw9_3;

%=========================================================================
%= Test hw9_1

% Get 6 points.
```

```
% Fit a quadratic by hw9_1.
nPoints     = size(points, 1);
step        = floor(nPoints ./ 6);
test_pt     = points(1:step:nPoints, :);
coef_1      = hw9_1(test_pt(1:6, :));
Z_quadratic = quadratic(X, Y, coef_1);

% Compute MSE at mesh grid.
% Draw fitted quadratic surface
mse_Z = compareResult(X, Y, Z_quadratic, Z_ellipsoidal);
title(['quadratic (fitted by 6 points, mse = ', ...
    num2str(mse_Z), ')']);
print -djpeg hw9_1;

%=========================================================================
%= Test hw9_2

% Fit a quadratic by hw9_2.
coef_2      = hw9_2(points);
Z_quadratic = quadratic(X, Y, coef_2);

% Compute MSE at mesh grid.
% Draw fitted quadratic surface
mse_Z = compareResult(X, Y, Z_quadratic, Z_ellipsoidal);
title(['quadratic (fitted by n > 6 points, mse = ', ...
    num2str(mse_Z), ')']);
print -djpeg hw9_2;

%=========================================================================
%= Test hw9_4

% Fit a quadratic by hw9_2.
coef_4     = hw9_4(points);
Z_quadratic = quadratic(X, Y, coef_4);

% Compute MSE at mesh grid.
% Draw fitted quadratic surface
mse_Z = compareResult(X, Y, Z_quadratic, Z_ellipsoidal);
title(['quadratic (fitted by n > 6 points w/ \sigma_i\in[0, 0.2], mse = ',
...
    num2str(mse_Z), ')']);
print -djpeg hw9_4;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function f = ellipsoidal(x, y)
% Syntax: f = ellipsoidal(x, y)
%
%         x, y   - inputs of the function
%         f      - function value
%
% Description: Give ellipsoidal function value at inputs
%
% Author: Haiying Liu
%   Date: Nov. 4, 2000
%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

dbstop if error

msg = nargchk(2, 2, nargin);
if (~isempty(msg))
  error(strcat('ERROR:', msg));
end

clear msg;

%=========================================================================

f = sqrt((x - 2) .* (x - 2) + (y - 3) .* (y - 3)) + 2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function f = quadratic(x, y, coef)
% Syntax: f = quadratic(x, y, coef)
%
%         x, y   - inputs in format [x1, y1; x2, y2; ... xn, yn]
%         coef   - coefficient of the quadratic function
%         f      - function value in format [f1; f2; ... fn]
%
% Description: Give quadratic function value at inputs
%
% Author: Haiying Liu
%   Date: Nov. 4, 2000
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

dbstop if error

msg = nargchk(3, 3, nargin);
if (~isempty(msg))
  error(strcat('ERROR:', msg));
end

if length(coef) ~= 6
  error('The ''coef'' must have 6 components.');
end

clear msg;

%=========================================================================

f = coef(1) + coef(2) .* x + coef(3) .* y + ...
  coef(4) .* x .* y + coef(5) .* x .* x + coef(6) .* y .* y;

function mse_Z = compareResult(X, Y, Z_quadratic, Z_ellipsoidal)
% Syntax: mse_Z = compareResult(X, Y, Z_quadratic, Z_ellipsoidal)
%
% Description: Compare result
%
% Author: Haiying Liu
%   Date: Nov. 4, 2000
```

```
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

dbstop if error

msg = nargchk(4, 4, nargin);
if (~isempty(msg))
  error(strcat('ERROR:', msg));
end

clear msg;

global AZ;
global EL;

%=========================================================================

% Compute MSE at mesh grid.
diff_Z      = Z_quadratic(:) - Z_ellipsoidal(:);
sqrt_diff_Z = diff_Z .* diff_Z;
mse_Z       = mean(sqrt_diff_Z);

% Draw fitted quadratic surface
figure;
mesh(X, Y, Z_quadratic);
xlabel('x');
ylabel('y');
zlabel('z');
view(AZ, EL);
```

- hw9_4.m

```
function coef = hw9_4(points)
% Syntax: coef = hw9_4(points)
%
%         points - n points known on the plane in the format
%                  [x1, y1, f1; x2, y2, f2; ...; xn, yn, fn]
%         coef   - coefficient (format [a0, ..., a5] for
%                  a0 + a1 * x + a2 * y + a3 * xy + a4x^2 + a5y^2
%
% Description: CMSC828D HW9_2
%
% Author: Haiying Liu
%   Date: Nov. 4, 2000
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

dbstop if error

msg = nargchk(1, 1, nargin);
if (~isempty(msg))
  error(strcat('ERROR:', msg));
end

clear msg;
```

```
%====================================================================

% Compose the linear equations in the format Ax = b.
nPoints = size(points, 1);
if nPoints < 6
  error('ERROR: The number of points must be larger than 6.');
end

A = zeros(nPoints, 6);
b = zeros(nPoints, 1);

for index = 1:nPoints
  x = points(index, 1);
  y = points(index, 2);
  A(index, :) = [1, x, y, x * y, x * x, y * y];
  f = points(index, 3);
  b(index)    = f;
end

% Solve the equations if they are not degenerated.
% distance = || Ax - b ||
% Minimize distance.
stdDev = abs(rand(size(A, 1)) * 0.2);
coef   = solve_by_svd(A, b, stdDev);
```

- hw9_5.m

```
function coef = hw9_5
% Syntax: coef = hw9_5
%
% Description: CMSC828D HW9_5
%
% Author: Haiying Liu
%   Date: Nov. 5, 2000
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

dbstop if error

msg = nargchk(0, 0, nargin);
if (~isempty(msg))
  error(strcat('ERROR:', msg));
end

clear msg;

%====================================================================

% Initialization.
halfWndSize = 2;
nTestPoints = 8;
maxD        = 11;
yR          = 15;

% Read left and right images.
```

```
imageL  = imread('imageL.tif');
imageL  = double(imageL);

imageR  = imread('imageR.tif');
imageR  = double(imageR);

% Randomly select one point (x, 15) in row 15 of right image
% used as reference image.
nRow    = size(imageL, 1);
nCol    = size(imageL, 2);

ptStep  = round((nCol - maxD - maxD) / (nTestPoints - 1));

xR_test = maxD:ptStep:nCol - maxD;

for index = 1:length(xR_test)

  xR = xR_test(index);

  % Check the SSD and C for (x + d, 15) in left image
  % for all 0 <= d <= maxD.
  yL     = yR;
  wndR   = imageR(yR - halfWndSize:yR + halfWndSize, ...
    xR - halfWndSize:xR + halfWndSize); % 5 x 5 window

  SSD    = zeros(maxD + 1, 1);
  C      = zeros(maxD + 1, 1);

  for d = 0:maxD
    xL = xR + d;
    wndL = imageL(yL - halfWndSize:yL + halfWndSize, ...
      xL - halfWndSize:xL + halfWndSize); % 5 x 5 window
    [SSD(d + 1), C(d + 1)] = score(wndL, wndR);
  end

  % Plot d-SSD and d-C curves
  figure;

  subplot(2, 2, 1);
  imshow(imageL, []);
  hold on;
  plot([xR, xR + d, xR + d, xR, xR], ...
    [yR - halfWndSize, yR - halfWndSize, yR + halfWndSize, ...
      yR + halfWndSize, yR - halfWndSize], 'g');
    title(['Left image (', num2str(xR), '~',num2str(xR + maxD), ...
      ', ', num2str(yR), ')']);

  subplot(2, 2, 2);
  imshow(imageR, []);
  hold on;
  plot(xR, yR, 'g+');
  title(['Right image (', num2str(xR), ', ', num2str(yR), ')']);

  showLeft   = xR-10;
  showRight  = xR + maxD + 10;
  showUp     = yR - 10;
  showBottom = yR + 10;
```

```
  subplot(2, 2, 3);
  plot([1:maxD + 1], SSD);
  xlabel('d');
  ylabel('SSD');
  title('SSD');

  subplot(2, 2, 4);
  plot([1:maxD + 1], C);
  xlabel('d');
  ylabel('C');
  title('Cross Correlation');

  print('-djpeg ', ['hw9_5_', num2str(index)]);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [SSD, C] = score(wndL, wndR)
% Syntax: [SSD, C] = score(wndL, wndR)
%
% Description: Compute
%                SSD (sum of sqared differences) and
%                C (cross correlation)
%              of two candidate matching window
%
% Author: Haiying Liu
%   Date: Nov. 5, 2000
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

dbstop if error

msg = nargchk(2, 2, nargin);
if (~isempty(msg))
  error(strcat('ERROR:', msg));
end

clear msg;

if size(wndL, 1) ~= size(wndR, 1) | ...
    size(wndL, 2) ~= size(wndR, 2)
  error('Candidate windows must have same size.');
end

%=========================================================================

nRow            = size(wndL, 1);
nCol            = size(wndL, 2);

SSD             = 0;
C_numerator     = 0;
C_denominator_1 = 0;
C_denominator_2 = 0;

for row = 1:nRow
```

```
  for col = 1:nCol
    wndL_ij = wndL(row, col);
    wndR_ij = wndR(row, col);
    SSD = SSD + (wndL_ij - wndR_ij) * (wndL_ij - wndR_ij);

    C_numerator = C_numerator + wndL_ij * wndR_ij;
    C_denominator_1 = C_denominator_1 + wndL_ij * wndL_ij;
    C_denominator_2 = C_denominator_2 + wndR_ij * wndR_ij;
  end
end

C = C_numerator / sqrt(C_denominator_1 * C_denominator_2);
```

- solve_by_svd.m

```
function x = solve_by_svd(A, b, stddev)
% Syntax: x = solve_by_svd(A, b, stddev)
%
% Description: Solve linear system Ax = b by SVD.
%
% Author: Haiying Liu
%   Date: Nov. 4, 2000
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

dbstop if error

msg = nargchk(2, 3, nargin);
if (~isempty(msg))
  error(strcat('ERROR:', msg));
end

clear msg;

%=========================================================================

% Arguments checking.
nRow_A      = size(A, 1);
nRow_b      = length(b);

if nRow_A ~= nRow_b
  error('ERROR: ''A'' must have same number of rows as ''b''');
end

if nargin < 3
  stddev = ones(nRow_A, 1);
else
  nRow_stddev = length(stddev);
  if nRow_stddev ~= nRow_A
    error('ERROR: ''stddev'' must have same number of rows as ''A''.');
  end
end

% Regularize parameter.
for row = 1:nRow_A
  A(row, :) = A(row, :) ./ stddev(row);
```

```
  b(row   ) = b(row   ) ./ stddev(row);
end

% Solve the linear system by SVD.
[U, S, V]       = svd(A);

size_S          = size(S);
nSingularValues = min(size_S(:));

x = zeros(size(V, 1), 1);

if nargin < 3
  epsilon = zeros(min(size_S), 1);
end

for index = 1:nSingularValues
  singularValue = S(index, index);
  if  singularValue ==0
    break;
  end

  x = x + U(:, index)' * b * V(:, index) ./ singularValue;
end
```