CMSC 828D: Fundamentals of Computer Vision
Homework 5[1]

Instructors: Larry Davis, Ramani Duraiswami,
Daniel DeMenthon, and Yiannis Aloimonos
Solution based on homework submitted by Haiying Liu

Date: October 27, 2000

1. Generate a parameterization along this curve using chord lengths. …

Solution: The Matlab script and the three column vectors are listed in appendix. The $x(s)$ and $y(s)$ are plotted in [Figure 1].

2. Using the function `polyfit` and the three vectors …

Solution: The Matlab script and evaluations are listed in appendix. The $x(s)$ and $y(s)$ fitted by polynomial functions are shown in [Figure 2].

3. Read chapter 3.3 of *Numerical recipes*. …

Solution: Done.

4. Fit cubic spines through the same data using the function `spline` …

Solution: The Matlab script and evaluations are listed in appendix. The $x(s)$ and $y(s)$ fitted by polynomial functions are shown in [Figure 3].

5. You can evaluate the derivatives …

Solution: The Matlab scripts and evaluations are listed in appendix. The figures are shown in [Figure 4]. From both figures and data, we can see the numerical result of $\frac{dy}{dx} = \frac{dy/ds}{dx/ds}$ is almost the same as the analytical expression.

6. Write a function that applies the Roberts operators to a given image …

Solution: The Matlab scripts are listed in appendix. The figures are shown in [Figure 5], including horizontal and vertical components of the gradient. Please note that the gradient directly from Roberts operators are 135° and 225° counter-clock-wise from the x-axis and needed to be rotated back to get the correct direction. The gaussian filter is normalized so that the sum of all entries is one. The output of function `my_gradient` in `hw5_6.m` includes the gradient magnitude, direction, and a mask whose non-zero entries are those pixels with maximum gradient magnitudes along their gradient directions. The mask is obtained by the function `nonMaximaSuppress.m`, which uses 3×3 window to compute the local maxima. The linear interpolation is used to get more-accurate result. Please see the Matlab scripts for detail.

[1] The e-version of this homework solution is posted at http://www.cfar.umd.edu/~hyliu/CMSC828D/hw5.zip.

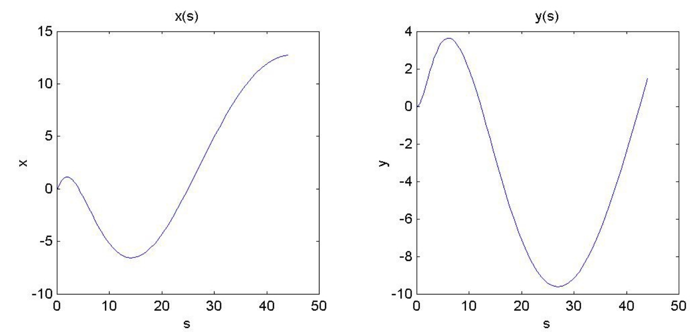


Figure 1: Original function $x(s)$ and $y(s)$.

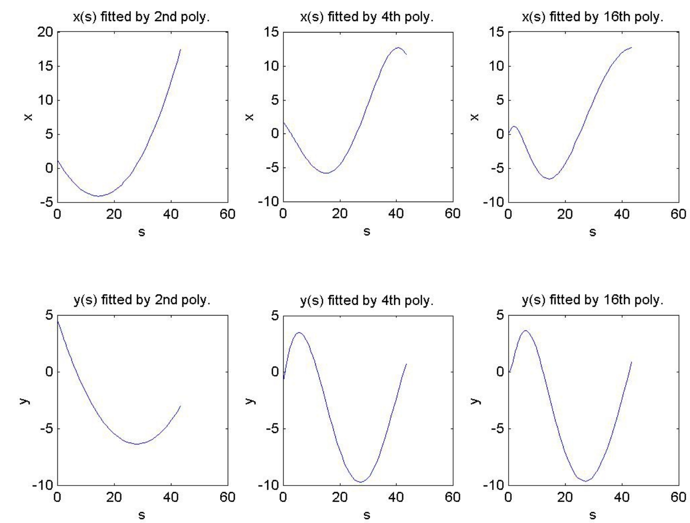


Figure 2: The $x(s)$ and $y(s)$ fitted by polynomial functions.

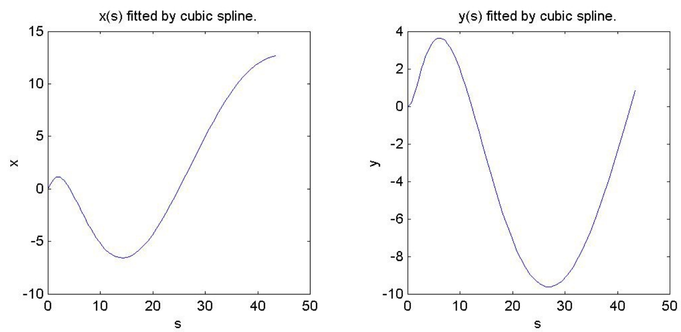


Figure 3: The $x(s)$ and $y(s)$ fitted by cubic spline.

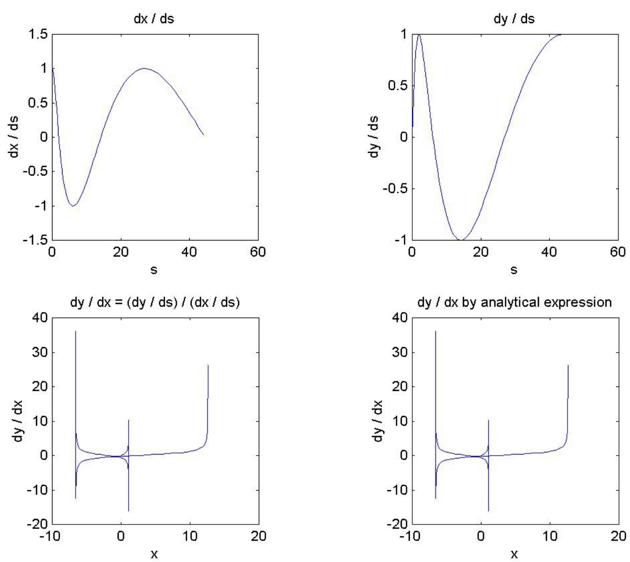
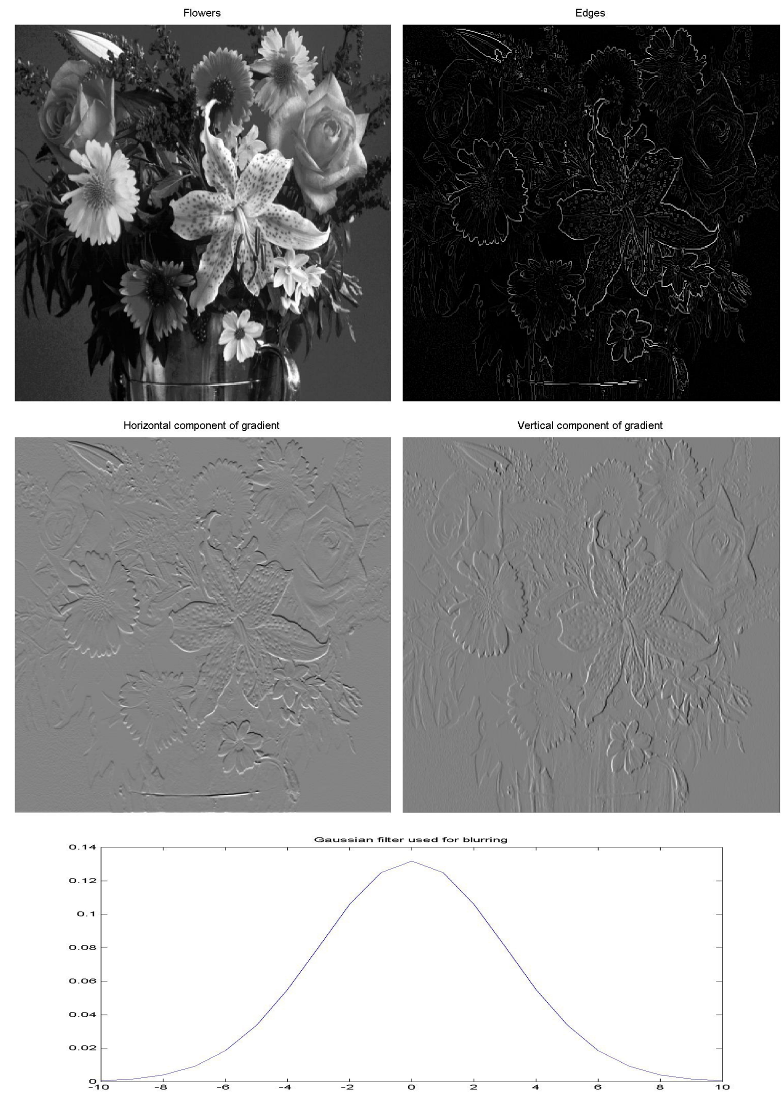


Figure 4: Derivatives $\frac{dy}{dx}$ by $\frac{dy}{dx} = \frac{dy/ds}{dx/ds}$ and by analytical expression.



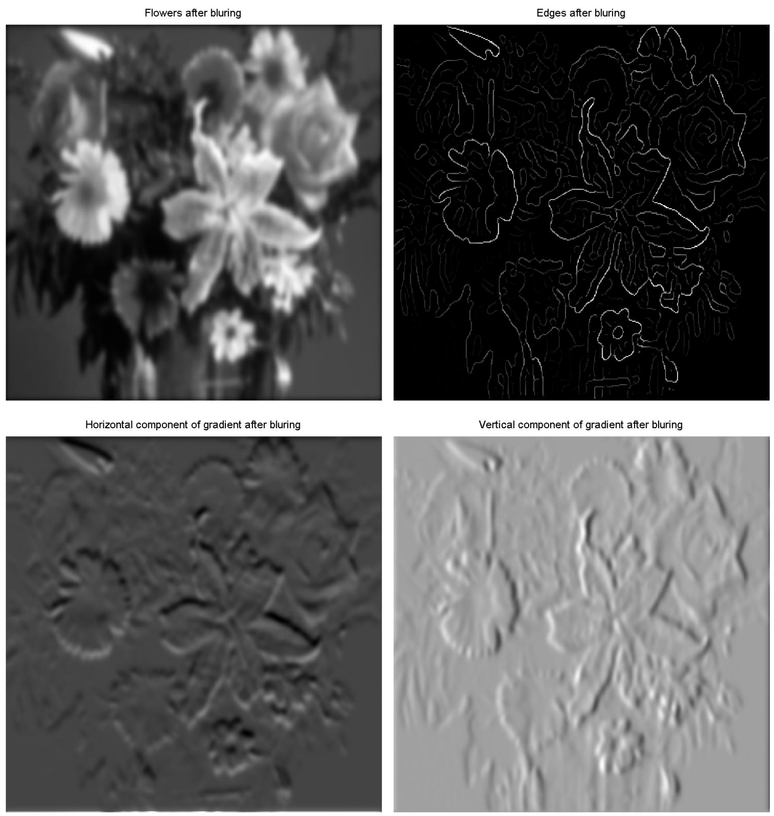


Figure 5: Images and their edges by Roberts operators.

CMSC 828D Fundamentals of Computer Vision Haiying Liu

Appendix:

- **hw5_1to5.m:**

```
function hw5_1to5
% Syntax: hw5_1to5
%
% Description: CMSC828D HW5_1
%
% Author: Haiying Liu
%   Date: Oct. 2, 2000
%

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

dbstop if error

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

clear msg;

%=====================================================================
%= Turn on the diary to save the result.

diary off;

filename = 'hw5_1to5.txt';
if (exist(filename, 'file'))
  delete(filename);
end

eval(['diary ', filename]);

%=====================================================================
%= Generate the Archimedean spiral.

from  = 0;
to    = 6.4;
step  = 0.1;
theta = [from:step:to]';
r = 2 .* theta;

x = r .* cos(theta);
y = r .* sin(theta);

figure;
plot(x, y);
axis square;
title('Archimedean spiral');

print -djpeg archimedeanSpiral;

%=====================================================================
%= Generate a parameterization along the curve.
```

```
disp('::::::::::::::');
disp(':: Problem 1 ::');
disp('::::::::::::::');
disp(' ');

nPoints = length(x);
s       = zeros(nPoints, 1);
s(1)    = 0;
for k = 2:nPoints
  s(k) = s(k - 1) + sqrt((x(k) - x(k - 1)) .^ 2 + (y(k) - y(k - 1)) .^ 2);
end

figure;

disp('------------------');
disp(' ');
disp('Original function');
s
x
y

subplot(1, 2, 1);
plot(s, x);
axis square;
xlabel('s');
ylabel('x');
title('x(s)');

subplot(1, 2, 2);
plot(s, y);
axis square;
xlabel('s');
ylabel('y');
title('y(s)');

print -djpeg hw5_1

%======================================================================
%= Fit the curve by 2nd, 4th, and 16th polynomial functions.

disp('::::::::::::::');
disp(':: Problem 2 ::');
disp('::::::::::::::');
disp(' ');

order    = [2, 4, 16];
nOrder   = length(order);

figure;

for index = 1:nOrder
  N = order(index);

  [poly_x, struct_x] = polyfit(s, x, N);
  [poly_y, struct_y] = polyfit(s, y, N);
```



```
  s_mid = midpoints(s);
  fit_poly_x = polyval(poly_x, s_mid);
  fit_poly_y = polyval(poly_y, s_mid);

  disp(' ');
  disp('------------------');
  disp(' ');
  disp(['Fitted by the polynomial function in the order of ', num2str(N)]);
  s_mid
  fit_poly_x
  fit_poly_y

  subplot(2, nOrder, index);
  plot(s_mid, fit_poly_x);
  axis square;
  xlabel('s');
  ylabel('x');
  if N == 2
    title(['x(s) fitted by ', num2str(N), 'nd poly.']);
  else
    title(['x(s) fitted by ', num2str(N), 'th poly.']);
  end

  subplot(2, nOrder, nOrder + index);
  plot(s_mid, fit_poly_y);
  axis square;
  xlabel('s');
  ylabel('y');
  if N == 2
    title(['y(s) fitted by ', num2str(N), 'nd poly.']);
  else
    title(['y(s) fitted by ', num2str(N), 'th poly.']);
  end
end

print -djpeg hw5_2

%======================================================================
%= Fit the curve by cubic spline functions.

disp('::::::::::::::');
disp(':: Problem 4 ::');
disp('::::::::::::::');
disp(' ');

pp_x = spline(s, x);
pp_y = spline(s, y);

fit_spline_x = ppval(pp_x, s_mid);
fit_spline_y = ppval(pp_y, s_mid);

disp(' ');
disp('------------------');
disp(' ');
disp('Fitted by cubic spline');
s_mid
fit_spline_x
```

```
fit_spline_y

figure;

subplot(1, 2, 1);
plot(s_mid, fit_spline_x);
axis square;
xlabel('s');
ylabel('x');
title(['x(s) fitted by cubic spline.']);

subplot(1, 2, 2);
plot(s_mid, fit_spline_y);
axis square;
xlabel('s');
ylabel('y');
title(['y(s) fitted by cubic spline.']);

print -djpeg hw5_4

%======================================================================
%= Evaluate dx/ds, dy/ds, dy/dx.

disp('::::::::::::::');
disp(':: Problem 5 ::');
disp('::::::::::::::');
disp(' ');

% Evaluate dy/dx by dy/ds, dx/ds.
dpp_x = diffpp(pp_x);
dpp_y = diffpp(pp_y);

fit_dpp_x = ppval(dpp_x, s);
fit_dpp_y = ppval(dpp_y, s);

dy_by_dx = fit_dpp_y ./ fit_dpp_x;

% Evaluate dy/dx analytically.
% Please note that x(1) = y(1) = 0, it is ignored.
nPoints = length(x);
for index = 2:nPoints
  xi = x(index);
  yi = y(index);
  norm                       = sqrt(xi * xi + yi * yi);
  tan_half_norm              = tan(0.5 * norm);
  dy_by_dx_analytical(index) = -(2 * tan_half_norm * norm + xi * xi + ...
    tan_half_norm * tan_half_norm * xi * xi) / ...
  (-2 * norm + xi * yi + xi * tan_half_norm * tan_half_norm * yi);
end

% Display result.
disp('x:');
x
disp('dy/dx by (dy/ds)/(dx/ds): ');
dy_by_dx
disp('dy/dx by analytical expression: ');
dy_by_dx_analytical'
```



```
% Draw the result.
figure;

subplot(2, 2, 1);
plot(s, fit_dpp_x);
axis square;
xlabel('s');
ylabel('dx / ds');
title('dx / ds');

subplot(2, 2, 2);
plot(s, fit_dpp_y);
axis square;
xlabel('s');
ylabel('dy / ds');
title('dy / ds');

subplot(2, 2, 3);
plot(x, dy_by_dx);
axis square;
xlabel('x');
ylabel('dy / dx');
title('dy / dx = (dy / ds) / (dx / ds)');

subplot(2, 2, 4);
plot(x, dy_by_dx_analytical);
axis square;
xlabel('x');
ylabel('dy / dx');
title('dy / dx by analytical expression');

print -djpeg hw5_5

%======================================================================
%= Stop recording.

diary off;

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

function s_mid = midpoints(s)
% Syntax: s_mid = midpoints(s)
%
% Description: Return the midpoints of 's'.
%
% Author: Haiying Liu
%   Date: Oct. 2, 2000
%

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

nPoints = length(s);
s_mid   = zeros(size(s));
s_mid   = s_mid(1:nPoints - 1);

for index = 1:nPoints - 1
```

```
  s_mid(index) = (s(index) + s(index + 1)) ./ 2;
end
```

- **diffpp.m:**

```
function dpp = diffpp(pp)
% DIFFPP differentiate a pp function
%
%        dpp = diffpp(pp)
%
% returns the first derivative of the spline in  PP .

% 2 oct 95 cb

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

[breaks,coefs] = unmkpp(pp);
[l,k] = size(coefs);
if k==1,
   dpp = mkpp(breaks,zeros(1,l));
else
   [k-1:-1:1];
   ans(ones(1,l),:).*coefs(:,1:k-1);
   dpp = mkpp(breaks,ans(:));
end
```

- **Result of hw5_1to5.m**

```
::::::::::::::
:: Problem 1 ::
::::::::::::::

------------------

Original function

s =

         0
    0.2000
    0.4020
    0.6079
    0.8195
    1.0386
    1.2666
    1.5049
    1.7547
    2.0169
    2.2926
    2.5823
    2.8869
    3.2068
    3.5426
    3.8946
    4.2633
    4.6489
    5.0518
    5.4721
    5.9102
    6.3661
    6.8400
    7.3322
    7.8427
    8.3717
    8.9192
    9.4854
   10.0703
   10.6741
   11.2967
   11.9384
   12.5990
   13.2788
   13.9777
   14.6957
   15.4330
   16.1895
   16.9654
   17.7606
   18.5751
   19.4090
   20.2624
   21.1352
   22.0275
   22.9393
   23.8706
   24.8214
   25.7918
   26.7817
   27.7913
   28.8204
   29.8692
   30.9376
   32.0256
   33.1334
   34.2607
   35.4078
   36.5745
   37.7610
   38.9671
   40.1930
   41.4386
   42.7040
   43.9891


x =

         0
    0.1990
    0.3920
    0.5732
    0.7368
```



```
    0.8776
    0.9904
    1.0708
    1.1147
    1.1189
    1.0806
    0.9979
    0.8697
    0.6955
    0.4759
    0.2122
   -0.0934
   -0.4381
   -0.8179
   -1.2285
   -1.6646
   -2.1204
   -2.5894
   -3.0649
   -3.5395
   -4.0057
   -4.4558
   -4.8820
   -5.2764
   -5.6316
   -5.9400
   -6.1946
   -6.3891
   -6.5174
   -6.5742
   -6.5552
   -6.4567
   -6.2759
   -6.0114
   -5.6623
   -5.2291
   -4.7136
   -4.1182
   -3.4469
   -2.7045
   -1.8972
   -1.0318
   -0.1165
    0.8400
    1.8278
    2.8366
    3.8554
    4.8726
    5.8764
    6.8547
    7.7954
    8.6863
    9.5157
   10.2720
   10.9442
   11.5220
   11.9959
   12.3571
   12.5982
   12.7128


y =

         0
    0.0200
    0.0795
    0.1773
    0.3115
    0.4794
    0.6776
    0.9019
    1.1478
    1.4100
    1.6829
    1.9607
    2.2369
    2.5053
    2.7593
    2.9925
    3.1986
    3.3717
    3.5059
    3.5959
    3.6372
    3.6255
    3.5574
    3.4302
    3.2422
    2.9924
    2.6806
    2.3079
    1.8759
    1.3876
    0.8467
    0.2578
   -0.3736
   -1.0411
   -1.7377
   -2.4555
   -3.1861
   -3.9208
   -4.6501
   -5.3646
   -6.0544
   -6.7099
   -7.3212
   -7.8790
   -8.3741
   -8.7978
   -9.1420
   -9.3993
   -9.5632
   -9.6280
   -9.5892
   -9.4433
   -9.1879
   -8.8220
   -8.3459
   -7.7609
   -7.0702
   -6.2778
   -5.3894
   -4.4117
   -3.3530
   -2.2224
   -1.0303
    0.2119
    1.4918

::::::::::::::
:: Problem 2 ::
::::::::::::::


------------------

Fitted by the
polynomial function
in the order of 2

s_mid =

    0.1000
    0.3010
    0.5049
    0.7137
    0.9291
    1.1526
    1.3858
    1.6298
    1.8858
    2.1548
    2.4375
    2.7346
    3.0469
    3.3747
    3.7186
    4.0790
    4.4561
    4.8504
    5.2620
    5.6911
    6.1381
    6.6031
    7.0861
    7.5875
    8.1072
    8.6454
    9.2023
    9.7779
```

```
   10.3722
   10.9854
   11.6176
   12.2687
   12.9389
   13.6282
   14.3367
   15.0643
   15.8113
   16.5775
   17.3630
   18.1678
   18.9921
   19.8357
   20.6988
   21.5813
   22.4834
   23.4049
   24.3460
   25.3066
   26.2868
   27.2865
   28.3059
   29.3448
   30.4034
   31.4816
   32.5795
   33.6970
   34.8343
   35.9912
   37.1678
   38.3641
   39.5801
   40.8158
   42.0713
   43.3465

fit_poly_x =

    1.1621
    1.0151
    0.8679
    0.7196
    0.5689
    0.4150
    0.2572
    0.0950
   -0.0718
   -0.2434
   -0.4198
   -0.6008
   -0.7861
   -0.9752
   -1.1677
   -1.3629
   -1.5600
```

```
   -1.7582
   -1.9566
   -2.1543
   -2.3500
   -2.5428
   -2.7312
   -2.9142
   -3.0901
   -3.2578
   -3.4155
   -3.5618
   -3.6950
   -3.8134
   -3.9152
   -3.9986
   -4.0617
   -4.1025
   -4.1189
   -4.1090
   -4.0705
   -4.0012
   -3.8988
   -3.7610
   -3.5854
   -3.3694
   -3.1107
   -2.8065
   -2.4542
   -2.0511
   -1.5944
   -1.0813
   -0.5088
    0.1260
    0.8261
    1.5946
    2.4348
    3.3497
    4.3427
    5.4171
    6.5764
    7.8239
    9.1633
   10.5980
   12.1318
   13.7683
   15.5113
   17.3647

fit_poly_y =

    4.5367
    4.3801
    4.2223
    4.0620
    3.8980
    3.7291
```

```
    3.5544
    3.3733
    3.1850
    2.9892
    2.7856
    2.5740
    2.3543
    2.1266
    1.8910
    1.6476
    1.3969
    1.1390
    0.8745
    0.6037
    0.3273
    0.0456
   -0.2405
   -0.5306
   -0.8239
   -1.1196
   -1.4169
   -1.7151
   -2.0132
   -2.3104
   -2.6057
   -2.8981
   -3.1867
   -3.4703
   -3.7478
   -4.0182
   -4.2803
   -4.5329
   -4.7746
   -5.0044
   -5.2208
   -5.4225
   -5.6082
   -5.7764
   -5.9257
   -6.0546
   -6.1616
   -6.2451
   -6.3037
   -6.3355
   -6.3391
   -6.3127
   -6.2545
   -6.1629
   -6.0361
   -5.8722
   -5.6694
   -5.4257
   -5.1394
   -4.8083
   -4.4306
   -4.0041
   -3.5269
```



```
   -2.9968

------------------

Fitted by the
polynomial function
in the order of 4

s_mid =

    0.1000
    0.3010
    0.5049
    0.7137
    0.9291
    1.1526
    1.3858
    1.6298
    1.8858
    2.1548
    2.4375
    2.7346
    3.0469
    3.3747
    3.7186
    4.0790
    4.4561
    4.8504
    5.2620
    5.6911
    6.1381
    6.6031
    7.0861
    7.5875
    8.1072
    8.6454
    9.2023
    9.7779
   10.3722
   10.9854
   11.6176
   12.2687
   12.9389
   13.6282
   14.3367
   15.0643
   15.8113
   16.5775
   17.3630
   18.1678
   18.9921
   19.8357
   20.6988
   21.5813
   22.4834
   23.4049
```

```
   24.3460
   25.3066
   26.2868
   27.2865
   28.3059
   29.3448
   30.4034
   31.4816
   32.5795
   33.6970
   34.8343
   35.9912
   37.1678
   38.3641
   39.5801
   40.8158
   42.0713
   43.3465

fit_poly_x =

    1.7124
    1.5943
    1.4717
    1.3436
    1.2088
    1.0664
    0.9153
    0.7546
    0.5835
    0.4014
    0.2076
    0.0019
   -0.2161
   -0.4464
   -0.6890
   -0.9434
   -1.2093
   -1.4859
   -1.7722
   -2.0669
   -2.3686
   -2.6754
   -2.9852
   -3.2957
   -3.6041
   -3.9074
   -4.2022
   -4.4848
   -4.7512
   -4.9973
   -5.2185
   -5.4099
   -5.5666
   -5.6835
   -5.7553
```

```
   -5.7766
   -5.7421
   -5.6466
   -5.4851
   -5.2527
   -4.9452
   -4.5585
   -4.0896
   -3.5361
   -2.8966
   -2.1709
   -1.3602
   -0.4673
    0.5031
    1.5442
    2.6467
    3.7988
    4.9855
    6.1883
    7.3850
    8.5494
    9.6502
   10.6513
   11.5112
   12.1818
   12.6086
   12.7299
   12.4759
   11.7683

fit_poly_y =

   -0.7710
   -0.4304
   -0.1018
    0.2173
    0.5285
    0.8326
    1.1297
    1.4192
    1.7000
    1.9702
    2.2278
    2.4701
    2.6944
    2.8975
    3.0763
    3.2275
    3.3477
    3.4337
    3.4824
    3.4909
    3.4564
    3.3765
    3.2492
    3.0728
```

```
    2.8462
    2.5686
    2.2400
    1.8609
    1.4324
    0.9562
    0.4350
   -0.1281
   -0.7292
   -1.3635
   -2.0255
   -2.7092
   -3.4075
   -4.1130
   -4.8175
   -5.5121
   -6.1875
   -6.8340
   -7.4415
   -7.9997
   -8.4982
   -8.9266
   -9.2748
   -9.5331
   -9.6923
   -9.7442
   -9.6815
   -9.4987
   -9.1914
   -8.7576
   -8.1976
   -7.5145
   -6.7145
   -5.8075
   -4.8074
   -3.7329
   -2.6079
   -1.4620
   -0.3314
    0.7409

Warning: Matrix is
close to singular or
badly scaled.
         Results may
be inaccurate. RCOND
= 1.578650e-027.
> In
G:\Programs\Matlab\to
olbox\matlab\polyfun\
polyfit.m at line 52
  In
G:\Course\CMSC828D\HW
5\hw5_1to5.m at line
108

Warning: Matrix is
close to singular or
badly scaled.
         Results may
be inaccurate. RCOND
= 1.578650e-027.
> In
G:\Programs\Matlab\to
olbox\matlab\polyfun\
polyfit.m at line 52
  In
G:\Course\CMSC828D\HW
5\hw5_1to5.m at line
109

------------------

Fitted by the
polynomial function
in the order of 16

s_mid =

    0.1000
    0.3010
    0.5049
    0.7137
    0.9291
    1.1526
    1.3858
    1.6298
    1.8858
    2.1548
    2.4375
    2.7346
    3.0469
    3.3747
    3.7186
    4.0790
    4.4561
    4.8504
    5.2620
    5.6911
    6.1381
    6.6031
    7.0861
    7.5875
    8.1072
    8.6454
    9.2023
    9.7779
   10.3722
   10.9854
   11.6176
   12.2687
   12.9389
   13.6282
   14.3367
   15.0643
   15.8113
   16.5775
   17.3630
   18.1678
   18.9921
   19.8357
   20.6988
   21.5813
   22.4834
   23.4049
   24.3460
   25.3066
   26.2868
   27.2865
   28.3059
   29.3448
   30.4034
   31.4816
   32.5795
   33.6970
   34.8343
   35.9912
   37.1678
   38.3641
   39.5801
   40.8158
   42.0713
   43.3465


fit_poly_x =

    0.0996
    0.2998
    0.4881
    0.6597
    0.8105
    0.9362
    1.0325
    1.0952
    1.1206
    1.1051
    1.0459
    0.9411
    0.7898
    0.5920
    0.3489
    0.0624
   -0.2644
   -0.6280
   -1.0240
   -1.4476
   -1.8934
   -2.3555
   -2.8277
```



```
   -3.3030
   -3.7743
   -4.2338
   -4.6737
   -5.0858
   -5.4621
   -5.7950
   -6.0769
   -6.3014
   -6.4623
   -6.5543
   -6.5729
   -6.5142
   -6.3751
   -6.1532
   -5.8471
   -5.4562
   -4.9815
   -4.4248
   -3.7898
   -3.0813
   -2.3050
   -1.4678
   -0.5771
    0.3590
    1.3317
    2.3315
    3.3475
    4.3681
    5.3806
    6.3727
    7.3323
    8.2484
    9.1101
    9.9057
   10.6223
   11.2472
   11.7717
   12.1915
   12.4963
   12.6645


fit_poly_y =

    0.0037
    0.0402
    0.1210
    0.2406
    0.3946
    0.5792
    0.7907
    1.0253
    1.2786
    1.5457
    1.8212
    2.0989
    2.3725
    2.6353
    2.8806
    3.1016
    3.2922
    3.4463
    3.5584
    3.6237
    3.6379
    3.5975
    3.4995
    3.3418
    3.1230
    2.8425
    2.5005
    2.0981
    1.6377
    1.1224
    0.5564
   -0.0551
   -0.7060
   -1.3894
   -2.0977
   -2.8228
   -3.5562
   -4.2888
   -5.0114
   -5.7145
   -6.3883
   -7.0232
   -7.6092
   -8.1369
   -8.5972
   -8.9817
   -9.2826
   -9.4932
   -9.6078
   -9.6212
   -9.5295
   -9.3293
   -9.0187
   -8.5970
   -8.0653
   -7.4264
   -6.6842
   -5.8436
   -4.9100
   -3.8899
   -2.7929
   -1.6309
   -0.4135
    0.8536

::::::::::::::
:: Problem 4 ::
::::::::::::::

------------------

Fitted by cubic
spline

s_mid =

    0.1000
    0.3010
    0.5049
    0.7137
    0.9291
    1.1526
    1.3858
    1.6298
    1.8858
    2.1548
    2.4375
    2.7346
    3.0469
    3.3747
    3.7186
    4.0790
    4.4561
    4.8504
    5.2620
    5.6911
    6.1381
    6.6031
    7.0861
    7.5875
    8.1072
    8.6454
    9.2023
    9.7779
   10.3722
   10.9854
   11.6176
   12.2687
   12.9389
   13.6282
   14.3367
   15.0643
   15.8113
   16.5775
   17.3630
   18.1678
   18.9921
   19.8357
   20.6988
   21.5813
   22.4834
   23.4049
   24.3460
   25.3066
   26.2868
```

```
   27.2865
   28.3059
   29.3448
   30.4034
   31.4816
   32.5795
   33.6970
   34.8343
   35.9912
   37.1678
   38.3641
   39.5801
   40.8158
   42.0713
   43.3465

fit_spline_x =

    0.1000
    0.2970
    0.4850
    0.6582
    0.8111
    0.9384
    1.0354
    1.0978
    1.1220
    1.1050
    1.0444
    0.9387
    0.7872
    0.5900
    0.3479
    0.0627
   -0.2630
   -0.6259
   -1.0218
   -1.4459
   -1.8926
   -2.3559
   -2.8290
   -3.3049
   -3.7762
   -4.2352
   -4.6741
   -5.0852
   -5.4608
   -5.7932
   -6.0754
   -6.3005
   -6.4623
   -6.5552
   -6.5744
   -6.5157
   -6.3761
   -6.1534
```

```
   -5.8463
   -5.4549
   -4.9800
   -4.4239
   -3.7899
   -3.0822
   -2.3064
   -1.4690
   -0.5775
    0.3596
    1.3331
    2.3327
    3.3479
    4.3673
    5.3792
    6.3717
    7.3326
    8.2498
    9.1112
    9.9053
   10.6207
   11.2467
   11.7735
   12.1918
   12.4936
   12.6719


fit_spline_y =

    0.0050
    0.0450
    0.1240
    0.2406
    0.3923
    0.5760
    0.7880
    1.0239
    1.2788
    1.5472
    1.8233
    2.1011
    2.3741
    2.6360
    2.8802
    3.1005
    3.2906
    3.4447
    3.5572
    3.6232
    3.6381
    3.5983
    3.5007
    3.3431
    3.1240
    2.8429
    2.5002
```

```
    2.0974
    1.6367
    1.1214
    0.5558
   -0.0552
   -0.7056
   -1.3886
   -2.0968
   -2.8221
   -3.5559
   -4.2890
   -5.0120
   -5.7152
   -6.3890
   -7.0234
   -7.6090
   -8.1363
   -8.5965
   -8.9812
   -9.2826
   -9.4937
   -9.6085
   -9.6217
   -9.5295
   -9.3288
   -9.0180
   -8.5966
   -8.0655
   -7.4271
   -6.6847
   -5.8433
   -4.9092
   -3.8898
   -2.7938
   -1.6310
   -0.4123
    0.8504

::::::::::::::
:: Problem 5 ::
::::::::::::::

x:

x =

         0
    0.1990
    0.3920
    0.5732
    0.7368
    0.8776
    0.9904
    1.0708
    1.1147
    1.1189
    1.0806
```



```
    0.9979
    0.8697
    0.6955
    0.4759
    0.2122
   -0.0934
   -0.4381
   -0.8179
   -1.2285
   -1.6646
   -2.1204
   -2.5894
   -3.0649
   -3.5395
   -4.0057
   -4.4558
   -4.8820
   -5.2764
   -5.6316
   -5.9400
   -6.1946
   -6.3891
   -6.5174
   -6.5742
   -6.5552
   -6.4567
   -6.2759
   -6.0114
   -5.6623
   -5.2291
   -4.7136
   -4.1182
   -3.4469
   -2.7045
   -1.8972
   -1.0318
   -0.1165
    0.8400
    1.8278
    2.8366
    3.8554
    4.8726
    5.8764
    6.8547
    7.7954
    8.6863
    9.5157
   10.2720
   10.9442
   11.5220
   11.9959
   12.3571
   12.5982
   12.7128

dy/dx by
(dy/ds)/(dx/ds):
```

```
dy_by_dx =

   -0.0020
    0.2029
    0.4195
    0.6717
    0.9902
    1.4395
    2.1782
    3.7578
   10.3775
  -16.1056
   -4.5882
   -2.6393
   -1.8078
   -1.3311
   -1.0114
   -0.7742
   -0.5851
   -0.4258
   -0.2853
   -0.1565
   -0.0345
    0.0850
    0.2054
    0.3304
    0.4640
    0.6113
    0.7794
    0.9784
    1.2250
    1.5477
    2.0015
    2.7089
    4.0085
    7.3169
   36.1640
  -12.4566
   -5.2718
   -3.2975
   -2.3581
   -1.7987
   -1.4204
   -1.1421
   -0.9244
   -0.7459
   -0.5938
   -0.4599
   -0.3386
   -0.2258
   -0.1183
   -0.0137
    0.0905
    0.1965
    0.3067
    0.4241
```

```
    0.5522
    0.6956
    0.8611
    1.0588
    1.3048
    1.6271
    2.0790
    2.7767
    4.0324
    7.0629
   26.3075

dy/dx by analytical
expression:

ans =

         0
    0.2024
    0.4197
    0.6717
    0.9903
    1.4395
    2.1783
    3.7580
   10.3786
  -16.1035
   -4.5880
   -2.6392
   -1.8078
   -1.3311
   -1.0114
   -0.7742
   -0.5851
   -0.4258
   -0.2853
   -0.1565
   -0.0345
    0.0850
    0.2054
    0.3304
    0.4640
    0.6113
    0.7794
    0.9784
    1.2250
    1.5477
    2.0015
    2.7089
    4.0085
    7.3170
   36.1649
  -12.4565
   -5.2718
   -3.2975
   -2.3581
   -1.7987
```

```
   -1.4204                 -0.0137                1.3048
   -1.1421                  0.0905                1.6271
   -0.9244                  0.1965                2.0790
   -0.7459                  0.3067                2.7767
   -0.5938                  0.4241                4.0322
   -0.4599                  0.5522                7.0653
   -0.3386                  0.6956               26.1776
   -0.2258                  0.8611
   -0.1183                  1.0588
```

- **hw5_6.m:**

```
function hw5_6
% Syntax: hw5_6
%
% Description: CMSC828D HW5_1
%
% Author: Haiying Liu
%   Date: Oct. 3, 2000
%

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

dbstop if error

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

clear msg;

%=======================================================================
%= Turn on the diary to save the result.

diary off;

filename = 'hw5_6.txt';
if (exist(filename, 'file'))
  delete(filename);
end

eval(['diary ', filename]);

%=======================================================================
%= Read image data.

% Read image from file.
image = imread('flowers.tif');

% Convert the color image into gray scale image.
image = rgb2gray(image);

% Convert the data type from uint8 to double.
image = double(image);
```



```
%=======================================================================
%= Find edges of original image.

% Apply roberts operators.
gradient  = my_gradient(image);

% Show result.
figure;
imshow(image, []);
axis square;
title('Flowers');
print -djpeg hw5_6_flower;

figure;
imshow(gradient.mag .* gradient.mask, []);
axis square;
title(['Edges']);
print -djpeg hw5_6_edge;

figure;
imshow(gradient.mag .* cos(gradient.dir), []);
axis square;
title('Vertical component of gradient');
print -djpeg hw5_6_edge_v;

figure;
imshow(gradient.mag .* sin(gradient.dir), []);
axis square;
title('Horizontal component of gradient');
print -djpeg hw5_6_edge_h;

%=======================================================================
%= Find edges of blured image.

% Construct gaussian filter.
% Reference: edge.m

% Magic numbers
sigma          = 3;
GaussianDieOff = .0001;

% Design the gausian filters
pw    = 1:10; % possible widths
ssq   = sigma*sigma;
width = max(find(exp(-(pw.*pw)/(2*sigma*sigma))>GaussianDieOff));
if isempty(width)
  width = 1;  % the user entered a really small sigma
end
t   = (-width:width);
len = 2*width+1;
t3  = [t-.5; t; t+.5]; % We will average values at t-.5, t, t+.5
gau = sum(exp(-(t3.*t3)/(2*ssq))).'/(6*pi*ssq);  % the gaussian 1-d filter
gau = gau ./ sum(gau(:)); % normalized

figure;
plot(t, gau);
title('Gaussian filter used for blurring');
```

```
print -djpeg hw5_6_gaus;

% Blur the image
% Apply gaussian filter along x and y direcitons.
blurredImage = conv2(gau', gau, image, 'same');

figure;
imshow(blurredImage, []);
axis square;
title('Flowers after bluring');
print -djpeg hw5_6_flower_blur;

% Apply roberts operators.
gradient_blur  = my_gradient(blurredImage);

% Show result.
figure;
imshow(gradient_blur.mag .* gradient_blur.mask, []);
axis square;
title('Edges after bluring');
print -djpeg hw5_6_edge_blur;

figure;
imshow(gradient_blur.mag .* cos(gradient_blur.dir), []);
axis square;
title('Vertical component of gradient after bluring');
print -djpeg hw5_6_edge_blur_v;

figure;
imshow(gradient_blur.mag .* sin(gradient_blur.dir), []);
axis square;
title('Horizontal component of gradient after bluring');
print -djpeg hw5_6_edge_blur_h;

%====================================================================
%= Stop recording.

diary off;

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

function gradient = my_gradient(data)
% Syntax: gradient =  my_gradient(data)
%
%         data        - data to be computed
%         gradient    - a structure contains results:
%                         .mag  : magnitude of the gradient
%                                 larger than 'threshold'
%                         .dir  : direction of the gradient
%                         .mask : mask for local maxima
%                                 1 : is local maxima
%                                 0 : is not local maxima
%                       .mag .* .mask yields edges.
%
% Description: Compute gradient by roberts operators
%
% Author: Haiying Liu
```



```
%   Date: Oct. 3, 2000
%

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

dbstop if error

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

clear msg;

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

% Construct roberts operator.
op_roberts = [1, 0; 0, -1] / sqrt(2);

% Get gradient.
bx = filter2(op_roberts, data, 'same');
by = filter2(rot90(op_roberts), data, 'same');

% Compute magnitude and direction.
theta        = 135 * pi / 180;
dx           = bx .* cos(theta) - by .* sin(theta);
dy           = bx .* sin(theta) + by .* cos(theta);
grad         = dx + i * dy;

gradient.mag = abs(grad);
gradient.dir = angle(grad);

% Suppress non-maxima
gradient.mask = zeros(size(data));
nRow          = size(data, 1);
nCol          = size(data, 2);

for row = 2:nRow - 1
  for col = 2:nCol - 1
    wnd = grad(row - 1:row + 1, col - 1:col + 1);
    gradient.mask(row, col) = ...
      nonMaximaSuppress(real(wnd), imag(wnd));
  end
end
```

- **nonMaximaSuppress.m:**

```
function r = nonMaximalSuppress(gradX, gradY)
%      Syntax: r = nonMaximalSuppress(gradX, gradY)
%
%         Arg: gradX(Y)    - local 3x3 gradient matrix.
%              r           - boolean result.
%                              1: is local maxima
%                              0: is NOT local maxima
%
% Description: Determine if the center of "local" is the max gradient
```

```
% Reference  : Numerical Recipes
%
% Author: Haiying
% Date  : Oct 3, 2000

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

% Check arguments.

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

clear msg;

%======================================================================
%= For each pixel (except those on the image boundary), check if
%= the gradient magnitude is a local (3x3 neighbourhood) maxima in
%= the gradient orientation:

r = 0;

y = 2;
x = 2;

dx = gradX(y, x);
dy = gradY(y, x);

mag = sqrt(gradX .* gradX + gradY .* gradY);

if (dx ~= 0 | dy ~= 0)
  if (abs(dy) > abs(dx))
    ux = abs(dx) / abs(dy);
    uy = 1;

    gb = mag(y - 1, x);
    gd = mag(y + 1, x);

    if (sign(dx) ~= sign(dy))
      ga = mag(y - 1, x - 1);
      gc = mag(y + 1, x + 1);
    else
      ga = mag(y - 1, x + 1);
      gc = mag(y + 1, x - 1);
    end
  else
    ux = abs(dy) / abs(dx);
    uy = 1;

    gb = mag(y, x + 1);
    gd = mag(y, x - 1);

    if (sign(dx) ~= sign(dy))
      ga = mag(y + 1, x + 1);
      gc = mag(y - 1, x - 1);
    else
```



```
      ga = mag(y - 1, x + 1);
      gc = mag(y + 1, x - 1);
    end
  end

  g1 = (ux * ga) + (uy - ux) * gb;
  g2 = (ux * gc) + (uy - ux) * gd;
  g  = sqrt(dx * dx + dy * dy);

  if (g >= g1 & g >= g2)
    r = 1;
  else
    r = 0;
  end
end
```