Non Linear Programming: Homework 7

vishvAs vAsuki

April 4, 2010

1 Binary least squares

Theoretical part submitted handwritten.

1.1 Code

```
function binaryLeastSquaresExperiments()
noiseLevels = [0.5 1 2 3];
for i=1:numel(noiseLevels)
    [A, b, xhat] = dataGenerator(noiseLevels(i));
    evaluateHeuristics(A, b, xhat, noiseLevels(i));
    keyboard
end
end

function evaluateHeuristics(A, b, xhat, noiseLevel)
numSamples = 100;
methods = 'ABCD';
numMethods = length(methods);
xhatApprox = cell(numMethods);
xhatApprox{1} = leastSquaresSolution(A, b);
[Z, z] = convexRelaxation(A, b);
xhatApprox{2} = signConvexRelaxation(A, b, Z, z);
xhatApprox{3} = signRank1approxConvexRelaxation(A, b, Z,
    z);
xhatApprox{4} = signProbabilisticConvexRelaxation(A, b, Z
    , z, numSamples);

lowestSquaredError = getSquaredError(A, b, xhat);
fprintf(1, 'Noise level: %d, Rough Lower bound on squared
    error: %d\n', noiseLevel, lowestSquaredError);
for i = 1:numMethods
    fprintf(1, '%c: disagreement fraction: %d
        squaredError: %d \n', methods(i), getDisagreement(
```

```
        xhat, xhatApprox{i}), getSquaredError(A, b,
        xhatApprox{i}));
end
end

function [A, b, xhat] = dataGenerator(s)
%  s is the noise level
randn('state',0)
m = 50;
n = 40;
A = randn(m,n);
xhat = sign(randn(n,1));
b = A*xhat + s*randn(m,1);
end

function squaredError = getSquaredError(A, b, x)
squaredError = (A*x - b)'*(A*x - b);
end

function disagreement = getDisagreement(x1, x2)
n = numel(x1);
disagreement = sum(x1 ~= x2)/n;
end

function xhat_A = leastSquaresSolution(A, b)
x_ls = A\b;
xhat_A = sign(x_ls);
end

function [Z, z] = convexRelaxation(A, b)
[m, n] = size(A);
%  keyboard
cvx_begin sdp
variable Z(n, n) symmetric;
variable z(n);
minimize trace(A'*A*Z) - sum(2*b'*A*z) + sum(b'*b)
subject to
diag(Z) == ones(n,1);
[Z z; z' 1] >= 0;
cvx_end
end

function xhat_B = signConvexRelaxation(A, b, Z, z)
xhat_B = sign(z);
end
```

```
function xhat_C = signRank1approxConvexRelaxation(A, b, Z
    , z)
[m, n] = size(A);
[v_1, ew_1] = eigs([Z z; z' 1], 1);
xhat_C = sign(v_1(1:n));
end

function xhat_D = signProbabilisticConvexRelaxation(A, b,
    Z, z, numSamples)
[m, n] = size(A);
xhat_D = zeros(n ,1);
error = Inf;
for i=1:numSamples
    v_samp = z + sqrtm(Z - z*z') * randn(n, 1);
    x_samp = sign(v_samp);
    error_i = getSquaredError(A, b, v_samp);
    if(error_i < error)
        xhat_D = x_samp;
        error = error_i;
    end
end
end
```

1.2 Results

1.3 Results

```
Noise level: 5.000000e-01,Rough Lower bound on squared error: 1.732435e+01
A: disagreement fraction: 0 squaredError: 1.732435e+01
B: disagreement fraction: 0 squaredError: 1.732435e+01
C: disagreement fraction: 1 squaredError: 6.519655e+03
D: disagreement fraction: 0 squaredError: 1.732435e+01

Noise level: 1,Rough Lower bound on squared error: 6.929739e+01
A: disagreement fraction: 2.500000e-02 squaredError: 1.620505e+02
B: disagreement fraction: 0 squaredError: 6.929739e+01
C: disagreement fraction: 1 squaredError: 6.515046e+03
D: disagreement fraction: 0 squaredError: 6.929739e+01

Noise level: 2,Rough Lower bound on squared error: 2.771895e+02
A: disagreement fraction: 1.000000e-01 squaredError: 9.085323e+02
B: disagreement fraction: 0 squaredError: 2.771895e+02
C: disagreement fraction: 1 squaredError: 6.609775e+03
D: disagreement fraction: 2.500000e-02 squaredError: 3.664738e+02

Noise level: 3,Rough Lower bound on squared error: 6.236765e+02
A: disagreement fraction: 1.750000e-01 squaredError: 1.151512e+03
```

```
B: disagreement fraction: 2.500000e-02 squaredError: 6.736883e+02
C: disagreement fraction: 9.750000e-01 squaredError: 7.334321e+03
D: disagreement fraction: 1.000000e-01 squaredError: 8.187229e+02
```

2 Approximation with trigonometric polynomials

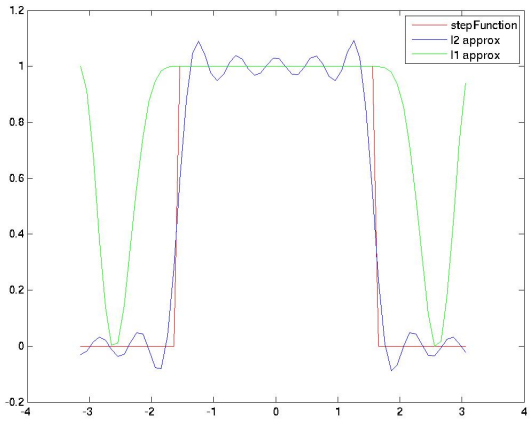
Theoretical part submitted handwritten.

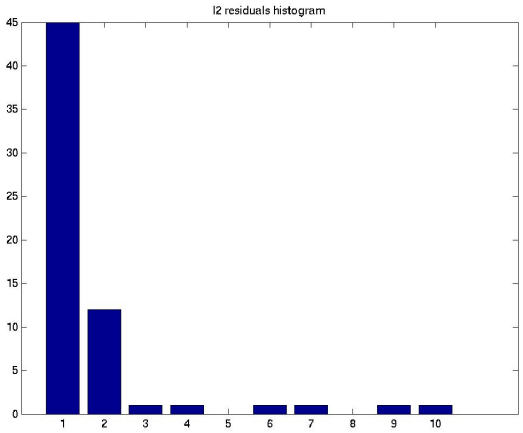
2.1 Code

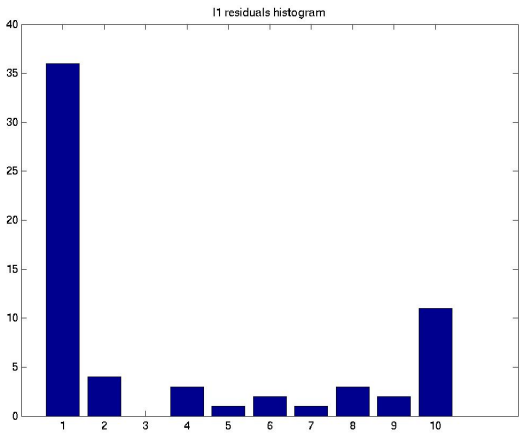
```
function trigonometricApproximationExperiments
inputPoints = (-pi:0.1:pi)';
stepFunctionValues = (abs(inputPoints) <= pi/2);
K = 10;

cosCoefficients_l2 = getL2ApproxCoefficients(K);
approximateValues_l2 = getApproximateValues(
    cosCoefficients_l2, inputPoints);

cosCoefficients_l1 = getL1ApproxCoefficients(K);
approximateValues_l1 = getApproximateValues(
    cosCoefficients_l1, inputPoints);

figureHandle = figure;
figureHandle = plot(inputPoints, stepFunctionValues, '-r'
    );
hold on;
figureHandle = plot(inputPoints, approximateValues_l2, '-
    b');
hold on;
figureHandle = plot(inputPoints, approximateValues_l1, '-
    g');
figureHandle = legend('stepFunction', 'l2 approx', 'l1
    approx');
saveas(figureHandle, ['/u/vvasuki/vishvas/work/
    optimization/hw/hw7/code/
    trigonometricApproximationExperiments.jpg'], 'jpg');
close all;

figureHandle = figure;
bar(hist(abs(approximateValues_l2 - stepFunctionValues)))
    ;
title('l2 residuals histogram');
```

```
saveas(figureHandle, ['/u/vvasuki/vishvas/work/
    optimization/hw/hw7/code/
    trigonometricApproximationExperimentsResidualsL2.jpg'
    ], 'jpg');

figureHandle = figure;
bar(hist(abs(approximateValues_l1 - stepFunctionValues)))
    ;
title('l1 residuals histogram');
saveas(figureHandle, ['/u/vvasuki/vishvas/work/
    optimization/hw/hw7/code/
    trigonometricApproximationExperimentsResidualsL1.jpg'
    ], 'jpg');
close all;

end

function cosCoefficients = getL2ApproxCoefficients(K)
cosCoefficients = zeros(K+1,1);
cosCoefficients(1) = 1/2;
for l=1:ceil(K/2)
% Ranging over odd k.
k = 2*l-1;
% Storing with an offset of 1.
index = k + 1;
cosCoefficients(index) = ((-1)^(l+1))*(2/(k*pi));
end
end

function cosCoefficients = getL1ApproxCoefficients(K)
inputPoints = (-pi:0.1:pi)';
numInputPoints = numel(inputPoints);
stepFunctionValues = (abs(inputPoints) <= pi/2);
A = zeros(numInputPoints, K+1);
for k = 0:K
    A(:, k+1) = cos(k*inputPoints);
end
stepFunctionValues = (inputPoints <= pi/2);
cvx_begin
variable cosCoefficients(K+1);
minimize sum(abs(A*cosCoefficients - stepFunctionValues))
    ;
cvx_end
% keyboard
end
```

```
function approximateValues = getApproximateValues(
    cosCoefficients, inputPoints)
numInputPoints = numel(inputPoints);
approximateValues = zeros(numInputPoints, 1);
for i = 1:numInputPoints
approximateValues(i) = getApproximation(cosCoefficients,
    inputPoints(i));
end
%  keyboard
end

function approximation = getApproximation(cosCoefficients
    , t)
K = numel(cosCoefficients) -1;
approximation = 0;
for k = 0:K
approximation = approximation + cosCoefficients(k+1)*cos(
    k*t);
end
end
```







l2 has better tracking: due to greater sensitivity to outliers.