Homework 4

Dustin Leatherman

April 26, 2020

Contents

1 Part 1

Make a Table. The first column of this table should contain values of m, the second column should be ϵ_{median}. Note that the median of ϵ depends on m.

Change the number of iterations to 20. For each value of m, (where $A \in R^{m \times 4m}$), run the program 5 times and record the median of ϵ. Do this for $m = 200, 190, 180, 170, 160, 150, 140, 130, 106, 105, 104, 103$

m	ϵ_{median}
200	1.90357e-09
190	4.83011e-09
180	5.91492e-08
170	3.16302e-08
160	4.70064e-08
150	1.56469e-06
140	1.07261e-07
130	6.72136e-08
106	371.846
105	251.155
104	477.818
103	998.845

```
%%% Iterative Hard Threshold algorithm %%%

mrows = [200, 190, 180, 170, 160, 150, 140, 130, 106, 105, 104, 103]
sparse = 9;
T = 20;

for m = mrows

  rows = m;
  N = 4 * rows;
  E = [];
  for i = 1:5
    Amat = gen_rand_mtx(rows, N);
    x_true = gen_x_true(N);

    yvect = Amat * x_true;    %%% y = Ax
    xvect = zeros(N, 1);

    for k = 1:T
      xvect = est_x(xvect, yvect, Amat, N, sparse);
    end

    is_conditioned(Amat);

    % [x_true([1:50],1)   xvect([1:50],1)];
```

```
    error = get_err(x_true, xvect);
    E = [E; error];
  end

  fprintf("Dimension: %i\n", m)
  fprintf("Median: %i\n", median(E))
end
```

2 Part 2

> Explore IHT with MatLab. Try changing the number of rows or columns of the matrix A. Try changing the 9-sparse vector to 15-spares vector. For example, can IHT reconstruct a 20-sparse vector with $A \in R^{200\times500}$? That is just one example, you can think of many more interesting questions.
>
> Describe carefully at least 2 interesting scenarios that you have tried. What did you observe?

2.1 Estimate Row Number

Davies describes a lower bound for $M \geq cslog(N/s)$ where c is some constant. When M meets this criteria, then "a random construction of A can achieve the RIP required... with high probability"(Davies, p6).

`https://arxiv.org/pdf/0805.0510.pdf`

Trying this with various $N = [300, 500, 1000, 2000, 4000, 8000, 16000]$ and various sparsity levels showed acceptable low median error amounts; however, only a 16-sparse is shown.

For s = 16,

N	Estimate M	Median Error
300	282	5.93634e-06
500	331	2.76957e-06
1000	397	5.85607e-09
2000	464	6.79957e-09
8000	597	3.85131e-10
16000	664	2.10875e-08

```
%%% Iterative Hard Threshold algorithm %%%
sparse = 16;
```

```
T = 50;
N_LIST = [300, 500, 1000, 2000, 8000, 16000];
% c gt 5. [Davies, Blumensath]
c = 6;
for N = N_LIST
    % based on Davies, Blumensath
    m = ceil(c * sparse * log(N / sparse));
    E = [];
    for i = 1:5
        Amat = gen_rand_mtx(m, N);
        x_true = gen_x_true(N);

        yvect = Amat * x_true;    %%% y = Ax
        xvect = zeros(N, 1);

        for k = 1:T
            xvect = est_x(xvect, yvect, Amat, N, sparse);
        end

        is_conditioned(Amat);

        % [x_true([1:50],1)   xvect([1:50],1)];

        error = get_err(x_true, xvect);
        E = [E; error];
    end

    fprintf("Dimension: %i\n", m)
    fprintf("Median: %i\n", median(E))
end
```

2.2 Estimate Iteration Number

Let y^s be the optimal s-sparse vector for y. The estimated error can be described by:

$$\epsilon_s = \|y - y^s\|_2 + \frac{1}{\sqrt{s}}\|y - y^s\|_1 + \|e\|_2$$

and the estimated iterations by

$$k^* = log_2(\frac{\|y^s\|_2}{\epsilon_s})$$

This formula gives negative numbers since ϵ_s is large, meaning that there is a lot of noise in the vector that y^s is not explaining. The absolute value of k is used instead since it is positive.

For s = 16,

N	Estimate M	Median Error
300	282	0.217618
500	331	0.198818
1000	397	0.129009
2000	464	0.113197
8000	597	0.089262
16000	664	0.0779062

The median error is greater than the values provided above but the amount of iterations and thus compute have significantly reduced with the median error still being in an acceptable range.

> Note: I do not understand why estimated algorithm for iterations "works". It is only producing negative values since the noise in the y vector is large.

```
%%% Iterative Hard Threshold algorithm %%%

sparse = 16;
N_LIST = [300, 500, 1000, 2000, 8000, 16000];
% c gt 5. [Davies, Blumensath]
c = 6;
for N = N_LIST
    % based on Davies, Blumensath
    m = ceil(c * sparse * log(N / sparse));
    E = [];
    for i = 1:5
        Amat = gen_rand_mtx(m, N);
        x_true = gen_x_true(N);

        yvect = Amat * x_true;    %%% y = Ax
        xvect = zeros(N, 1);
```

```
        % signal-to-noise ratio for sparse estimates
        % this value is almost always negative so I am not sure why this formula shoul
        est_iter = log2(norm(s_approx(yvect, sparse, m)) / est_err(yvect, sparse, m));
        fprintf('Estimated Iterations: %i\n', est_iter)
        % est_iter is usually a negative float so taking abs() and rounding up.
        for k = 1:ceil(abs(est_iter))
            xvect = est_x(xvect, yvect, Amat, N, sparse);
        end

        is_conditioned(Amat);

        % [x_true([1:50],1)   xvect([1:50],1)];

        error = get_err(x_true, xvect);
        E = [E; error];
    end

    fprintf('Dimension: %i\n', m)
    fprintf('Median: %i\n', median(E))
end
% Estimated Error [Davies, Blumensath]
% Assumes e is a unit vector. i.e. ||e||_2 = 1
% Large values indicate lots of noise.
function err = est_err(yvect, sparse, m)
   noise = yvect - s_approx(yvect, sparse, m);
   err = norm(noise) + (norm(noise, 1) / sqrt(sparse)) + 1;
end
```

3 Original Code

```
%%% Iterative Hard Threshold algorithm %%%

sparse = 9;
rows = 200;
N = 4 * rows;
T = 50;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%   Make the random matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
Amat = randn(rows,N);
Amat = 1 / sqrt(rows) * Amat;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%%%    Make the true x vector
%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x_true = zeros(N,1);
x_true(6) = 1.2;
x_true(7) = 0.7;
x_true(8) = 1.2;
x_true(15) = 0.7;
x_true(16) = -1.2;
x_true(17) = -0.7;
x_true(18) = -1.3;
x_true(19) = 1.3;
x_true(39) = 1.3;

yvect = Amat * x_true;    %%% y = Ax
xvect = zeros(N,1);

%%%      Here is how to use the sort function
%%% c = [1 3 5 7 9 2 4 6 8 10];
%%% [b, place] = sort(c, 'descend');
%%% c( place(1:4));

for k = 1:T
    uvect = xvect + Amat' * (yvect - Amat * xvect);
    [u_sort, place] = sort(abs(uvect), 'descend');

    xvect = zeros(N,1);

    % populates the xvect in descending order
    for j = 1:sparse
        xvect(place(j)) = uvect(place(j));
    end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%%% The following 3 lines check that matrix Amat is well-conditioned
%%%
%%% Think of e_max and e_min as (1+ delta_s) and (1 - delta_s) in RIP
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

A_submat = Amat(:, [39,15,16,17,18,19,6,7,8]);

eigenvalues = eig(A_submat' * A_submat);

e_max = max(eigenvalues);
e_min = min(eigenvalues);
[e_max, e_min]

%%% Compare the true values with the predicted values.
%%% The predicted values are in the vector xvect.
%%% Show the first 50 values of the true values and predicted values.

[x_true([1:50],1)   xvect([1:50],1)]

norm(x_true - xvect)/norm(x_true)
```

4 Refactored Code

```
%%% Iterative Hard Threshold algorithm %%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%   Make the random matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function Amat = gen_rand_mtx(rows, N)
  Amat = randn(rows,N);
  Amat = 1 / sqrt(rows) * Amat;
end

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

```
%%%
%%%    Make the true x vector
%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function x_true = gen_x_true(N)
  x_true = zeros(N,1);
  x_true(6) = 1.2;
  x_true(7) = 0.7;
  x_true(8) = 1.2;
  x_true(15) = 0.7;
  x_true(16) = -1.2;
  x_true(17) = -0.7;
  x_true(18) = -1.3;
  x_true(19) = 1.3;
  x_true(39) = 1.3;
end


%%%      Here is how to use the sort function
%%% c = [1 3 5 7 9 2 4 6 8 10];
%%% [b, place] = sort(c, 'descend');
%%% c( place(1:4));

function xvect = est_x(xvect, yvect, Amat, N, sparse)
  uvect = xvect + Amat' * (yvect - Amat * xvect);

  xvect = s_approx(uvect, sparse, N);
end

function v_sparse = s_approx(v, s, N)
  v_sparse = zeros(N, 1);
  [v_sort, place] = sort(abs(v), 'descend');
  % populates v_sparse in descending order
  for j = 1:s
      v_sparse(place(j)) = v(place(j));
  end
end

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

```
%%%
%%% The following 3 lines check that matrix Amat is well-conditioned
%%%
%%% Think of e_max and e_min as (1+ delta_s) and (1 - delta_s) in RIP
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [e_max, e_min] = is_conditioned(Amat)
  A_submat = Amat(:, [39,15,16,17,18,19,6,7,8]);

  eigenvalues = eig(A_submat' * A_submat);

  e_max = max(eigenvalues);
  e_min = min(eigenvalues);
end


%%% Compare the true values with the predicted values.
%%% The predicted values are in the vector xvect.
%%% Show the first 50 values of the true values and predicted values.


function error = get_err(x_true, xvect)
  error = norm(x_true - xvect)/norm(x_true);
end

%% Put it all together

sparse = 9;
rows = 100;
N = 4 * rows;
T = 50;

Amat = gen_rand_mtx(rows, N);
x_true = gen_x_true(N);

yvect = Amat * x_true;    %%% y = Ax
xvect = zeros(N, 1);

for k = 1:T
  xvect = est_x(xvect, yvect, Amat, N, sparse);
```

```
end

is_conditioned(Amat)

[x_true([1:50],1)   xvect([1:50],1)]

get_err(x_true, xvect)
```