On Some Regression Methods

A. AlMomani

For the purpose of this totorial, we will create a dataset using a 3-dimensional function of random variables;

```
clearvars; close all; clc;

rng(1)


N = 1000; %number of sampled points to use


% Creat the randome variables
x = randn(N,1);
y = randn(N,1);
z = randn(N,1);

f = 3 + 2*x -3*y +0.9*z.^2;

% Add wighte gaussian noise to the function
f = awgn(f,10);

% create the library of candidate functions
Phi = [ones(size(x)), x, y, z, ... %linear terms
     x.^2, x.*y, x.*z, y.^2, y.*z, z.^2]; % quadratic terms

d = size(Phi,2);% Regression dimension, or, number of candidate functions

% According to the order of A, the optimal solution should be
TS = [3, 2, -3, 0, 0, 0, 0, 0, 0, 0.9]';
```

Now, we state the problem as:

Given Φ and f, Find β that best satisfy:

$$\|\Phi\beta - f\|_2$$

Brute-Force Search and L_0 Minimization

The Brute Force search, also known as exhaustive search or generate and test, is a general problem-solving technique that systematically enumerates all possible candidates for the solution and checks whether each candidate satisfies the problem's statement.

In our problem, we are trying to find the optimal solution that minimize the objective function:

$$\min_{\beta} (\|\Phi\beta - f\|_2 + \lambda\|\beta\|_0)$$

where λ is a penalty value. See the function implementation below.

First, if we test the function with no penalty value, we find the solution:

```
xbest = bruteForce(Phi,f,0);
disp(xbest)
```

```
    3.0165
    2.0026
   -2.9879
    0.0151
   -0.0047
   -0.0101
    0.0089
   -0.0104
   -0.0003
    0.9043
```

which is clearly not a sparse solution. The reason is that the least-squares solution will invest in all possible candidate functions to minimize the -Norm of the fitting, even if in the micro-scale.

If we add a small penalty, λ, we will have:

```
xbest = bruteForce(Phi,f,0.01);
disp(xbest)
```

```
    3.0121
    2.0007
   -2.9885
    0.0150
         0
         0
         0
   -0.0108
         0
    0.9040
```

So, a good strategy is to choose a span of penalty values, and see how the solution will change:

```
lambda = [0, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100]';
B = zeros(d,length(lambda)); %to store the solution for each penaly value as columns

for i=1:length(lambda)
    xbest = bruteForce(Phi,f,lambda(i));
    B(:,i) = xbest;
end
disp(B)
```

```
    3.0165    3.0165    3.0165    3.0121    2.9991    2.9991    2.9991         0
    2.0026    2.0026    2.0026    2.0007    1.9995    1.9995    1.9995         0
   -2.9879   -2.9879   -2.9879   -2.9885   -2.9899   -2.9899   -2.9899         0
    0.0151    0.0151    0.0151    0.0150         0         0         0         0
   -0.0047   -0.0047   -0.0047         0         0         0         0         0
   -0.0101   -0.0101   -0.0101         0         0         0         0         0
    0.0089    0.0089    0.0089         0         0         0         0         0
   -0.0104   -0.0104   -0.0104   -0.0108         0         0         0         0
   -0.0003         0         0         0         0         0         0         0
    0.9043    0.9043    0.9043    0.9040    0.9044    0.9044    0.9044         0
```

From the left to the right, you see how as we increase the penalty value, we get a more sparse solution, until (with large penalty) we get a solution with all entries equal to zero.

Dimension - Fitting Trade-off

To visualize this trade of between the dimension of the solution ($\|\beta\|_0$) and the fitting quality ($\|\Phi\beta - f\|_2$), we can use a two-y-axis plot.

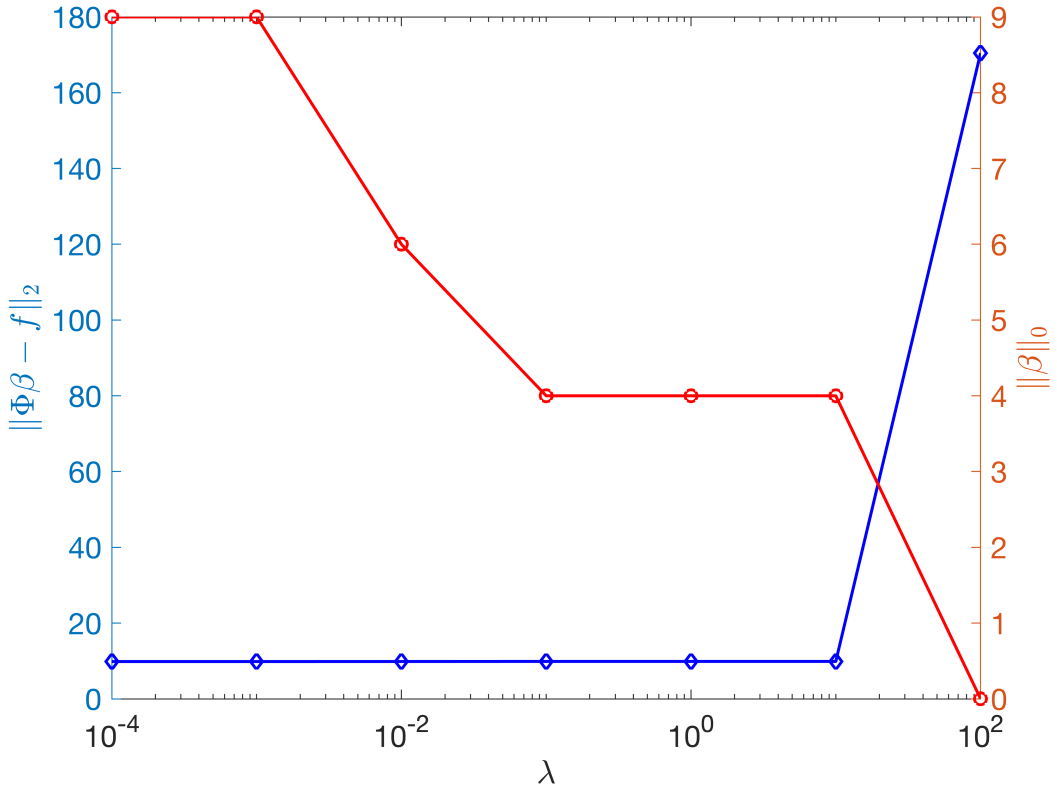
First, we compute these values for each solution:

```
L0 = sum(logical(B),1)'; %L0 for each solution
L2 = sqrt(sum((Phi*B-f).^2,1))'; %L2 of the fitting for each solution
```

Now we plot the two-y-axis plot as the following:

```
figure
yyaxis left
semilogx(lambda,L2,'-db','LineWidth',1.5)
hold on
ylabel("$\Vert \Phi\beta - f \Vert_2$","Interpreter","latex")

yyaxis right
semilogx(lambda,L0,'-or','LineWidth',1.5)
ylabel("$\Vert \beta \Vert_0$","Interpreter","latex")
xlabel("$\lambda$","Interpreter","latex")
set(gca,"FontSize",15)
```



It is VERY IMPORTANT to learn how to read this figure. We will discuss this in the class in detail.

For now, you can see the a penalty $\lambda = 10$, gave the most sparse solution with negligable increase in the error.

So, the best solution is (The true solution to the right for comparison, the solution we obtained from the Brute-Force L_0 regularization to the left)

```
disp([B(:,7), TS])
```

```
    2.9991    3.0000
    1.9995    2.0000
   -2.9899   -3.0000
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
    0.9044    0.9000
```

SINDy, or Sequential Least Squares

In SINDy, there is no penalty or regularization. The only measure of quality is the magnitude of the parameters, and it judges the relevance of the function by the magnitude of the parameter associated with that function.

The main idea of SINDy is to iterativily eliminate the low magnitue parameters by applying a hard-threshold function on the least squares solution as the following:

$$T(\beta, \lambda) = \begin{cases} \beta_{i,} & \forall |\beta_i| \geq \theta, i =, 1, \dots, N \\ 0, & \text{otherwise.} \end{cases}$$

where θ is a threshold parameter, and N is number of candidate functions.

```
theta = 0.1;
sindySolution = SINDy(Phi, f, theta);
disp([sindySolution, TS])
```

```
    2.9991    3.0000
    1.9995    2.0000
   -2.9899   -3.0000
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
    0.9044    0.9000
```

LASSO

Lasso (least absolute shrinkage and selection operator) is a regression analysis method that performs variable selection and regularization to enhance the prediction accuracy and interpretability of the resulting statistical model.

The main objective in lasso regression is to minimize the function:

$$\min_{\beta} \left(\alpha \|\Phi\beta - f\|_2^2 + \lambda \|\beta\|_1 \right)$$

There are many algorithms and methods to solve Lasso regression, see this reference here. However, the commonly used ones are:

- Coordinate Descent Algorithm
- ADMM Algorithm

Both algorithms are available in the built-in Matlab function "lasso"

Coordinate Descent Algorithm

Coordinate descent is based on the idea that the minimization of a multivariable function can be achieved by minimizing it along one direction at a time. In the simplest case of cyclic coordinate descent, one cyclically iterates through the directions, one at a time, minimizing the objective function with respect to each coordinate direction at a time.

Basic Algorithm:

Choose $\beta^0 \in \mathbb{R}^n$;

Set $k \leftarrow 0$;

repeat

Choose index i_k with uniform probability from $\{1, 2, \ldots, n\}$, independently of choices at prior iterations;

Set $\beta^{k+1} \leftarrow \beta^k - \alpha_k [\nabla f(\beta^k)]_{i_k} e_{i_k}$ for some $\alpha_k > 0$;

$k \leftarrow k + 1$;

until termination test satisfied;

```
[B, fitInfo] = lasso(Phi,f);
size(B)
```

```
ans = 1×2
    10   100
```

```
disp(B)
```

```
  Columns 1 through 12

         0         0         0         0         0         0         0         0         0         0
    2.0022    2.0022    2.0021    2.0021    2.0020    2.0020    2.0019    2.0018    2.0018    2.0017    2.
   -2.9876   -2.9876   -2.9876   -2.9876   -2.9875   -2.9875   -2.9874   -2.9874   -2.9874   -2.9873   -2.
    0.0148    0.0147    0.0147    0.0146    0.0146    0.0146    0.0145    0.0144    0.0144    0.0143    0.
   -0.0045   -0.0045   -0.0045   -0.0045   -0.0045   -0.0044   -0.0044   -0.0044   -0.0043   -0.0043   -0.
   -0.0097   -0.0097   -0.0096   -0.0096   -0.0095   -0.0095   -0.0094   -0.0094   -0.0093   -0.0092   -0.
    0.0085    0.0085    0.0084    0.0084    0.0083    0.0083    0.0082    0.0082    0.0081    0.0080    0.
   -0.0102   -0.0102   -0.0102   -0.0102   -0.0101   -0.0101   -0.0101   -0.0101   -0.0100   -0.0100   -0.
         0         0         0         0         0         0         0         0         0         0
    0.9040    0.9040    0.9040    0.9040    0.9039    0.9039    0.9039    0.9039    0.9038    0.9038    0.

  Columns 13 through 24

         0         0         0         0         0         0         0         0         0         0
    2.0014    2.0013    2.0011    2.0010    2.0008    2.0006    2.0005    2.0002    2.0000    1.9998    1.
   -2.9871   -2.9870   -2.9870   -2.9869   -2.9868   -2.9867   -2.9865   -2.9864   -2.9863   -2.9861   -2.
    0.0141    0.0140    0.0139    0.0137    0.0136    0.0135    0.0133    0.0131    0.0129    0.0127    0.
```

```
 -0.0042   -0.0041   -0.0040   -0.0040   -0.0039   -0.0038   -0.0037   -0.0036   -0.0035   -0.0034   -0.
 -0.0089   -0.0088   -0.0087   -0.0086   -0.0084   -0.0083   -0.0081   -0.0079   -0.0077   -0.0075   -0.
  0.0077    0.0076    0.0075    0.0073    0.0072    0.0070    0.0068    0.0066    0.0064    0.0062    0.
 -0.0099   -0.0098   -0.0097   -0.0097   -0.0096   -0.0095   -0.0094   -0.0094   -0.0093   -0.0091   -0.
       0         0         0         0         0         0         0         0         0         0
  0.9036    0.9035    0.9035    0.9034    0.9033    0.9032    0.9031    0.9030    0.9029    0.9028    0.

Columns 25 through 36

       0         0         0         0         0         0         0         0         0         0
  1.9988    1.9985    1.9981    1.9976    1.9971    1.9966    1.9960    1.9954    1.9947    1.9939    1.
 -2.9855   -2.9853   -2.9851   -2.9848   -2.9845   -2.9841   -2.9838   -2.9834   -2.9829   -2.9824   -2.
  0.0120    0.0117    0.0113    0.0110    0.0106    0.0101    0.0096    0.0091    0.0085    0.0079    0.
 -0.0030   -0.0028   -0.0026   -0.0024   -0.0021   -0.0019   -0.0016   -0.0013   -0.0010   -0.0006   -0.
 -0.0066   -0.0063   -0.0059   -0.0055   -0.0051   -0.0046   -0.0041   -0.0035   -0.0029   -0.0022   -0.
  0.0053    0.0049    0.0045    0.0041    0.0036    0.0031    0.0026    0.0019    0.0013    0.0005
 -0.0087   -0.0086   -0.0084   -0.0082   -0.0080   -0.0077   -0.0075   -0.0072   -0.0069   -0.0066   -0.
       0         0         0         0         0         0         0         0         0         0
  0.9023    0.9021    0.9019    0.9016    0.9014    0.9011    0.9008    0.9005    0.9001    0.8997    0.

Columns 37 through 48

       0         0         0         0         0         0         0         0         0         0
  1.9912    1.9902    1.9892    1.9881    1.9869    1.9855    1.9841    1.9825    1.9809    1.9791    1.
 -2.9806   -2.9799   -2.9790   -2.9781   -2.9771   -2.9760   -2.9748   -2.9735   -2.9720   -2.9702   -2.
  0.0055    0.0046    0.0036    0.0025    0.0013         0         0         0         0         0
       0         0         0         0         0         0         0         0         0         0
       0         0         0         0         0         0         0         0         0         0
       0         0         0         0         0         0         0         0         0         0
 -0.0053   -0.0048   -0.0042   -0.0036   -0.0029   -0.0021   -0.0012   -0.0003         0         0
       0         0         0         0         0         0         0         0         0         0
  0.8982    0.8977    0.8971    0.8964    0.8956    0.8948    0.8939    0.8929    0.8918    0.8906    0.

Columns 49 through 60

       0         0         0         0         0         0         0         0         0         0
  1.9725    1.9698    1.9670    1.9638    1.9603    1.9565    1.9523    1.9477    1.9426    1.9371    1.
 -2.9639   -2.9614   -2.9586   -2.9555   -2.9522   -2.9485   -2.9445   -2.9401   -2.9352   -2.9299   -2.
       0         0         0         0         0         0         0         0         0         0
       0         0         0         0         0         0         0         0         0         0
       0         0         0         0         0         0         0         0         0         0
       0         0         0         0         0         0         0         0         0         0
       0         0         0         0         0         0         0         0         0         0
       0         0         0         0         0         0         0         0         0         0
  0.8861    0.8843    0.8824    0.8802    0.8779    0.8753    0.8725    0.8693    0.8659    0.8622    0.

Columns 61 through 72

       0         0         0         0         0         0         0         0         0         0
  1.9170    1.9089    1.9001    1.8904    1.8798    1.8681    1.8553    1.8413    1.8258    1.8089    1.
 -2.9106   -2.9028   -2.8943   -2.8850   -2.8748   -2.8636   -2.8513   -2.8378   -2.8229   -2.8067   -2.
       0         0         0         0         0         0         0         0         0         0
       0         0         0         0         0         0         0         0         0         0
       0         0         0         0         0         0         0         0         0         0
       0         0         0         0         0         0         0         0         0         0
       0         0         0         0         0         0         0         0         0         0
       0         0         0         0         0         0         0         0         0         0
  0.8486    0.8431    0.8371    0.8306    0.8234    0.8155    0.8068    0.7973    0.7868    0.7754    0.

Columns 73 through 84

       0         0         0         0         0         0         0         0         0         0
  1.7475    1.7230    1.6960    1.6664    1.6339    1.5983    1.5592    1.5163    1.4691    1.4174    1.
 -2.7477   -2.7240   -2.6981   -2.6697   -2.6385   -2.6042   -2.5666   -2.5253   -2.4800   -2.4303   -2.
```

```
         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0
    0.7338    0.7172    0.6989    0.6789    0.6569    0.6328    0.6063    0.5772    0.5453    0.5103    0.

  Columns 85 through 96

         0         0         0         0         0         0         0         0         0         0
    1.2300    1.1550    1.0727    0.9823    0.8831    0.7743    0.6546    0.5205    0.3732    0.2116    0.
   -2.2502   -2.1781   -2.0989   -2.0121   -1.9167   -1.8121   -1.6972   -1.5699   -1.4302   -1.2769   -1.
         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0
    0.3834    0.3326    0.2768    0.2156    0.1485    0.0748         0         0         0         0

  Columns 97 through 100

         0         0         0         0
         0         0         0         0
   -0.7256   -0.5058   -0.2647         0
         0         0         0         0
         0         0         0         0
         0         0         0         0
         0         0         0         0
         0         0         0         0
         0         0         0         0
         0         0         0         0
```

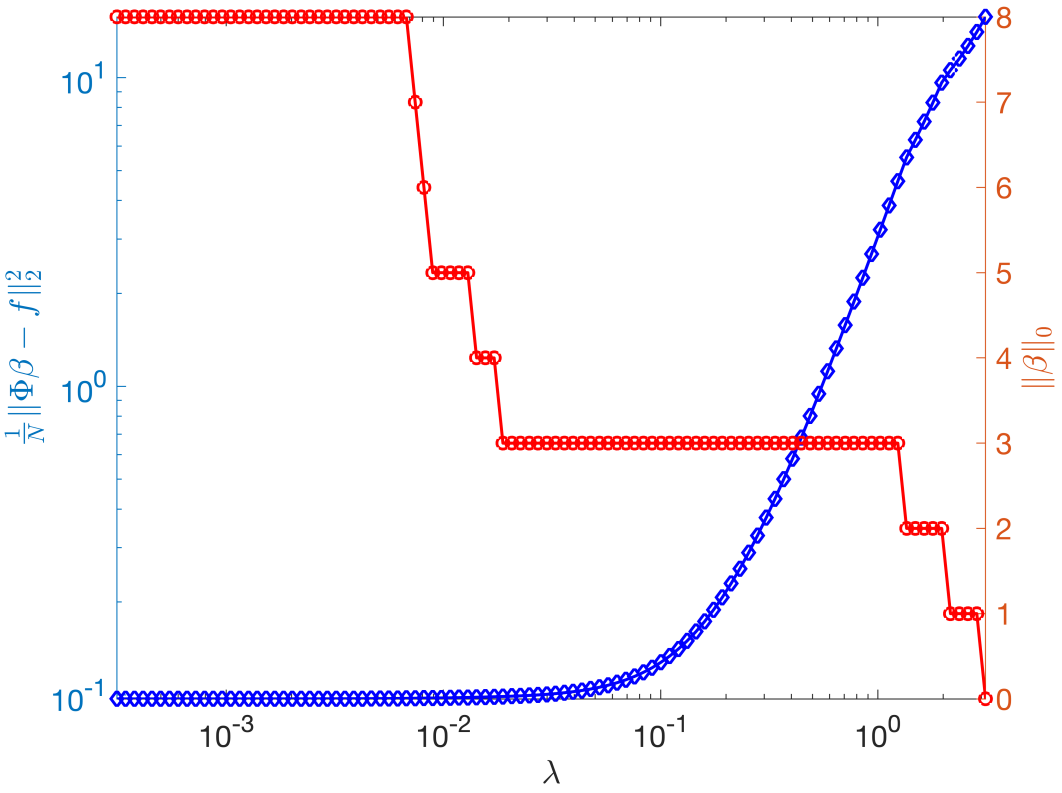
As you see, lasso creates a span of penalty values λ, and return the solutions matrix B where the i^{th} columns in B associated with λ_i. The fitInfor structure contain the fitting information.

```
lambda = fitInfo.Lambda; %penalty values used
L2     = fitInfo.MSE;    %The MSE for each solution
L0     = fitInfo.DF;     %the L0 norm of each solution
```

Now, we can create the trade-off curves similar to the example above (Brute-Force search).

```
figure
yyaxis left
loglog(lambda,L2,'-db','LineWidth',1.5)
hold on
ylabel("$\frac{1}{N}\Vert \Phi\beta - f \Vert_2^2$","Interpreter","latex")

yyaxis right
semilogx(lambda,L0,'-or','LineWidth',1.5)
ylabel("$\Vert \beta \Vert_0$","Interpreter","latex")
xlabel("$\lambda$","Interpreter","latex")
set(gca,"FontSize",15)
xlim([min(lambda) max(lambda)])
```



Hard-Iterative-Thresholding Trade-Off

You saw how lasso returns a set of possible solutions, where we can perform statistical analysis to pick an optimal solution out of them.

With this in mined, one may think of the hard thresholding, as in SINDy, as a tool to optain such set of solutions. Moreover, there will be no need then to select a thresholding parameter θ.

Simply, we iteratively eliminate the lowest magnitude parameter, until the last parameter. See "Hit" function below.

The result will look as the following:

```
B = Hit(Phi,f);
disp(B)
```

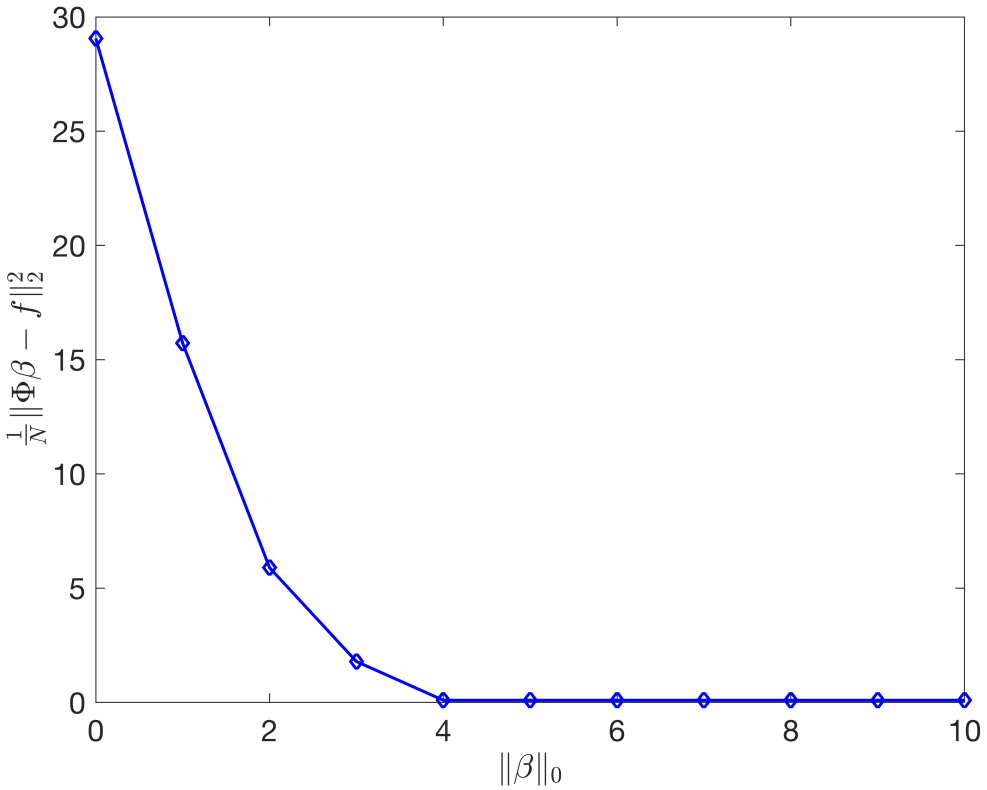
```
    3.0165    3.0165    3.0119    3.0115    3.0121    2.9999    2.9991    3.9182    3.8895    3.6521
    2.0026    2.0026    2.0027    2.0024    2.0007    2.0001    1.9995    2.0307         0         0
   -2.9879   -2.9879   -2.9883   -2.9882   -2.9885   -2.9902   -2.9899   -3.0027   -2.9794         0
    0.0151    0.0151    0.0150    0.0152    0.0150    0.0147         0         0         0         0
   -0.0047   -0.0047         0         0         0         0         0         0         0         0
   -0.0101   -0.0101   -0.0101   -0.0093         0         0         0         0         0         0
    0.0089    0.0089    0.0087         0         0         0         0         0         0         0
   -0.0104   -0.0104   -0.0106   -0.0104   -0.0108         0         0         0         0         0
   -0.0003         0         0         0         0         0         0         0         0         0
    0.9043    0.9043    0.9043    0.9042    0.9040    0.9042    0.9044         0         0         0
```

Again, we can compute the MSE and L_0-norm for each solution as the following:

```
L0 = sum(logical(B),1);
L2 = sum((Phi*B-f).^2,1)./length(f);
```

and since there was no tuning or thresholding parameter, then we can just plot the L_0 norm vs the MSE directly

```
figure
plot(L0,L2,'-db','LineWidth',1.5)
hold on
ylabel("$\frac{1}{N}\Vert \Phi\beta - f \Vert_2^2$","Interpreter","latex")
xlabel("$\Vert \beta \Vert_0$","Interpreter","latex")
set(gca,"FontSize",15)
```



Functions Inplementation

Brute Force

```
function xbest = bruteForce(A,y,penalty)

if (nargin<3) || isempty(penalty)
    penalty = 0;
end

d = size(A,2);
```

```
% Create all possible Boolean combinations of dimension N
P = logical(dec2bin(0:(2^d-1))-'0');

% Initialize the vector of the error
E = zeros(size(P,1),1);

% loop through all possible combinations
for i=1:size(P,1)
    B = A(:,P(i,:));

    % Find the cost (Objective function and penalty)
    E(i) = norm(B*pinv(B)*y-y) + penalty*sum(P(i,:));
end

% Find the index of the combination with the minimum cost
[~,ix] = min(E);


xbest = zeros(d,1);

xbest(P(ix,:)) = pinv(A(:,P(ix,:)))*y;


end
```

SINDy

This implementation is for teaching purposes. Refer to the author's paper (click here) for the code from the authors.

```
function beta = SINDy(A,y,theta)

    T = @(b,t) (abs(b)>t); %threshould function returns boolean indicator

    beta = pinv(A)*y;
    beta0  = zeros(size(beta));
    while sum(xor(~~beta,~~beta0)) %In the original code there is no such termination
                                   %however, this termination condition can
                                   %be more efficient in some cases.
                                   %
        beta0 = beta;              %Store the previous solution
        ix = T(beta,theta);        %Find the large parameters indicator

        beta(ix) = pinv(A(:,ix))*y; %Re-compute the large parameters
        beta(~ix)= 0;               %threshold the small parameters
    end

end
```

Hard-Iterative-Thresholding

```
function B = Hit(A,y)

x0 = pinv(A)*y;
N  = size(A,2);
B  = zeros(N,N+1);

IX = 1:N;
i = 1;
while ~isempty(IX)
    B(IX,i) = x0;
    [~,pos] = min(abs(x0));
    IX(pos) = [];
    x0 = pinv(A(:,IX))*y;
    i = i+1;
end
end
```