ALTERNATING DIRECTION METHOD OF MULTIPLIERS (ADMM)

DEEPTHI SUDHARSAN (CB.EN.U4AIE19022)

 B.TECH CSE-AI, CEN DEPARTMENT, AMRITA SCHOOL OF ENGINEERING

ACKNOWLEDGEMENT

This project would not have been possible without the support and guidance of Dr. K P Soman, who has provided me with sufficient knowledge and guidelines even during these unprecedented and trying times.

I would also like to thank our Computer Science and Engineering (Artificial Intelligence) department for giving me this opportunity to nurture and hone my skills.

Furthermore, I would like to thank the Amrita Vishwa Vidyapeetham management for ample resources to avail my project needs and a platform for online lectures during this lockdown.

TABLE OF CONTENTS

ABSTRACT4

HOW TO APPROACH [CONVEX] ADMM PROBLEMS? (TWO METHODS) 6

METHOD 16

METHOD 2 9

SOME EXAMPLES WHERE ADMM COMES INTO USE 10

CONVEX OPTIMIZATION PROBLEMS 10

LASSO10

LINEAR PROGRAMMING 12

QUADRATIC PROGRAMMING14

INTERSECTION OF POLYHEDRA 17

TOTAL VARIATION MINIMIZATION20

NON - CONVEX OPTIMIZATION PROBLEMS 23

REGRESSOR SELECTION (LASSO)23

OTHER NOTABLE APPLICATIONS OF ADMM27

REFERENCES28

ABSTRACT

**Optimization** is finding the best/optimum output by maximizing or minimizing a given function such that it also satisfies certain constraints. There are different types of optimization that are in use for a variety of purposes. **Convex and Non – convex optimizations** are two well-known types. The optimization is convex or non – convex based on whether or not its objective function or any of its constraints are convex or non-convex. Suppose, we are given the area of a lawn and we also know that it’s perimeter is as small as possible and we have to find the dimensions of the lawn. This is an example of optimization.

The alternating direction method of multipliers (**ADMM**) is a powerful iterative algorithm that solves convex (Linear programming, Quadratic Programming etc) optimization problems by breaking them into smaller pieces. It is a modification of the **augmented Lagrangian** that partially updates the dual variables over iterations.

Augmented lagrangian is a popular method for solving constrained optimization problems. Augmented lagragian method that uses partial updates is none other than ADMM. The classic ADMM splits a complex problem into subproblems which is easy to solve.

ADMM on nonlinear equality problems are difficult to solve as the nonlinear equality constraint makes subproblems nonconvex. Until now, the conditions to the existence of optimal values of these subproblems remain a mystery. When it comes to ADMM on non-convex optimization problems, ways to find approximating methods for nonconvex functions and inequalities are still unknown as the solution may not be the global optimal value but guarantees 90 to 95 % accurate value near to the global optimal value.

Keywords : *Alternating Direction Method of Multipliers (****ADMM****), Optimization, Augmented Lagrangian*

ALTERNATING DIRECTION METHOD OF MULTIPLIERS (ADMM)

# How to approach [convex] ADMM Problems? (Two Methods)

**Method 1:**





































**Let us consider the same minimization problem with different equality**

**constraint**



































We get the proximal algorithm equations









Now that we have a simplified model, we can start with augmented Lagrangian functions

of the form



With these simplified equations , we can easily write first two optimization equations

quite easily.

**Method 2:**

**Another form of ADMM is where the equality constraints actually lie over a region.**

**Let us look at an example,**



We convert it into



Where,

g(z) is the indicator function and



Augmented Lagrangian function can be written as









## Some Examples where ADMM comes into use

**CONVEX OPTIMIZATION EXAMPLES :**

1. **LASSO :**





**Code:**

Lasso function:

function x = lasso(A, b, lambda, rho)

MAX\_ITER = 10;

RELTOL = 1e-2; %error tolerance for ADMM

[m, n] = size(A);

Ata = A'\*A;

Atb = A'\*b;

x = zeros(n,1);

z = zeros(n,1);

u = zeros(n,1);

for k = 1:MAX\_ITER

% x-update

x\_1 = inv(Ata - rho\*eye(n)) \* (Atb - rho \* (z -u));

% z-update

z\_1 = (lambda/rho)\*(x\_1-(u/rho));

% u-update

u\_1 = u + RELTOL\*(x\_1 - z\_1);

z = z\_1;

x = x\_1;

u = u\_1;

end

end

Creating the problem and calling the function

m = 4; % number of examples

n = 5; % number of features

p = 0.05; % p = 0.05 is the sparsity density

x0 = sprandn(n,1,p);

A = randn(m,n);

A = A\*spdiags(1./sqrt(sum(A.^2))',0,n,n); % normalize

columns

b = A\*x0 + sqrt(0.001)\*randn(m,1);

lambda = 1;

x = lasso(A, b, lambda, 1.0)

Sample Output:

x =

1.0e+08 \*

-5.2996

-4.5646

3.3924

0.7834

2.9229

**NOTE: For small ‘n’ , we can use both “ relaxation and shrinkage method of solving “ and the above method without them for solving and getting the same answer. For large value of ‘n’, relaxation and shrinkage method of solving converges faster. Shrinkage will come if l1 norm there in the formulation. Relaxation is for faster convergence similar to the Cholesky decomposition for faster solution.**

**(2) Linear Programming (LP):**





**Code:**

LP function:

function z = linprog(c,A,b,rho,alpha)

MAX\_ITER = 10;

%rho is the augmented Lagrangian parameter.

%sol is returned in vector z

% alpha is the over-relaxation parameter(typical values

for alpha are

% between 1.0 and 1.8).

[m n] = size(A);

x = zeros(n,1);

z = zeros(n,1);

u = zeros(n,1);

for k =1:MAX\_ITER

%x-update

tmp = [rho\*eye(n), A';A, zeros(m)]\[rho\*(z-u)-c;b];

x = tmp(1:n);

%z-update with relaxation

zold = z;

x\_hat = alpha\*x+(1 - alpha)\*zold;

z = pos(x\_hat + u);

u = u + (x\_hat - z);

end

end

Creating the problem and calling the function:

n = 5; % dimension of x

m = 4; % number of equality constraints

c = rand(n,1) + 0.5; % create nonnegative price vector

with mean 1

x0 = abs(randn(n,1)); % create random solution vector

A = abs(randn(m,n)); % create random, nonnegative

matrix A

b = A\*x0;

linprog(c, A, b, 1.0, 1.0)

Sample Output:

ans =

0.5715

1.1079

1.4083

0.6878

0.6601

**(3) Quadratic Programming :**











**Code:**

QP function:

function x = qsolve1(P,q,A,b,rho,tolr,tols,iternum)

m=size(A,1);

n=size(P,1);

u=ones(n,1);

u(1,1)=0;

z=ones(n,1);

k=0;

nr=1; ns=1;

% convergence is controlled by norm nr of the primal

residual r

% and the norm ns of the dual residual s

while(( k <= iternum) && (ns > tols || nr < tolr))

z0=z;

k=k+1;

%KKT Matrix LHS

KK = [P + rho\*eye(n) A';A zeros(m,m)];

%KKT Matrix RHS

bb = [-q + rho\*(z-u); b];

%Solving KKT Matrix

xx=KK\bb;

%ADMM Updates

x = xx(1:n);

z = poslin(x+u); %Positive linear transfer

function

u = u + x - z;

% To test stopping criterion

r = x - z; %primal residual

nr = sqrt(r'\*r); %norm of primal residual

s = rho\*(z-z0); % dual residual

ns = sqrt(s'\*s); %norm of dual residual

end

end

Creating the problem and calling the function:

clc;

clear all;

close all;

P2 = [4 1 0 0;1 4 1 0;0 1 4 1;0 0 1 4];

q2 = [-4;-4;-4;-4];

A2 = [1 1 -1 0;1 -1 -1 0];

b2=[0;0];

MAX\_ITER = 100;

rho2 = 10;

tr = 10^(-12);

ts = 10^(-12);

Fvalue = qsolve1(P2,q2,A2,b2,rho2,tr,ts,MAX\_ITER);

xvalue = Fvalue(1)

yvalue = Fvalue(2);

zvalue = Fvalue(3);

tvalue = Fvalue(4);

Sample Output:

xvalue =

0.9032

**(4) Intersection of Polyhedral :**











**Code:**

Intersection of Polyhedral function:

function x = polyhedra\_intersection(A1, b1, A2, b2, alpha)

MAX\_ITER = 5;

n = size(A1,2);

x = zeros(n,1);

z = zeros(n,1);

u = zeros(n,1);

for k = 1:MAX\_ITER

% x-update

% use cvx to find point in first polyhedra

cvx\_begin quiet

variable x(n)

minimize (sum\_square(x - (z - u)))

subject to

A1\*x <= b1

cvx\_end

% z-update with relaxation

zold = z;

x\_hat = alpha\*x + (1 - alpha)\*zold;

% use cvx to find point in second polyhedra

cvx\_begin quiet

variable z(n)

minimize (sum\_square(x\_hat - (z - u)))

subject to

A2\*z <= b2

cvx\_end

u = u + (x\_hat - z);

end

end

Creating the problem and calling the function:

n = 5; % dimension of variable

m1 = 10; % number of faces for polyhedra 1

m2 = 12; % number of faces for polyhedra 2

c1 = 10\*randn(n,1); % center of polyhedra 1

c2 = -10\*randn(n,1); % center of polyhedra 2

% pick m1 n m2 random directions with different magnitudes for A1 n A2

% resp

% the value of resp b is found by traveling from the center along the normal

% vectors in resp A and taking its inner product with resp A.

A1 = diag(1 + rand(m1,1))\*randn(m1,n);

b1 = diag(A1\*(c1\*ones(1,m1) + A1'));

A2 = diag(1 + rand(m2,1))\*randn(m2,n);

b2 = diag(A2\*(c2\*ones(1,m2) + A2'));

% find the distance between the two polyhedra--make sure they overlap by

% checking if the distance is 0, if not expand A1 and A2 by a little more than half the

% distance

cvx\_begin quiet

variables x(n) y(n)

minimize sum\_square(x - y)

subject to

A1\*x <= b1

A2\*y <= b2

cvx\_end

if norm(x-y) > 1e-4

A1 = (1 + 0.5\*norm(x-y))\*A1;

A2 = (1 + 0.5\*norm(x-y))\*A2;

% recompute b's as appropriate

b1 = diag(A1\*(c1\*ones(1,m1) + A1'));

b2 = diag(A2\*(c2\*ones(1,m2) + A2'));

end

x = polyhedra\_intersection(A1, b1, A2, b2, 1.0)

Sample Output:

x =

1.0e-04 \*

0.1139

-0.1486

0.0945

-0.0014

-0.1195

**(5) Total Variation Minimization :**









**Code:**

Total Variation function:

function x = total\_variation(b, lambda, rho,alpha)

MAX\_ITER = 1000;

n = length(b);

e = ones(n,1);

D = spdiags([e -e], 0:1, n,n); %Sparse matrix formed from diagonals

x = zeros(n,1);

z = zeros(n,1);

u = zeros(n,1);

I = speye(n);

DtD = D'\*D;

for k = 1:MAX\_ITER

% x-update

x = (I + rho\*DtD) \ (b + rho\*D'\*(z-u));

% z-update with relaxation

zold = z;

Ax\_hat = alpha\*D\*x +(1-alpha)\*zold;

a = Ax\_hat + u;

kappa = lambda/rho;

z = max(0, a-kappa) - max(0, -a-kappa);

% u-update

u = u + Ax\_hat - z;

end

end

Creating the problem and calling the function:

% Total variation denoising with random data

clc;

clear all;

close all;

n = 2;

x0 = ones(n,1);

for j = 1:3

idx = randsample(n,1);

k = randsample(1:10,1);

x0(ceil(idx/2):idx) = k\*x0(ceil(idx/2):idx);

end

b = x0 + randn(n,1);

lambda = 5;

x = total\_variation(b, lambda, 1.0,1.0)

Sample Output:

x =

18.0913

2.7901

**NON - CONVEX OPTIMIZATION EXAMPLE:**

**(6) Regressor Selection (Non convex optimization) :**















**Code:**

Regressor Selection function (LASSO):

function x = regressor\_sel(A, b, K, rho)

[m, n] = size(A);

MAX\_ITER = 10;

% save a matrix-vector multiply

Atb = A'\*b;

x = zeros(n,1);

z = zeros(n,1);

u = zeros(n,1);

% cache the factorization

[L U] = factor(A, rho);

for k = 1:MAX\_ITER

% x-update

q = Atb + rho\*(z - u); % temporary value

if( m >= n ) % if skinny

x = U \ (L \ q);

else % if fat

x = q/rho - (A'\*(U \ ( L \ (A\*q) )))/rho^2;

end

% z-update with relaxation

zold = z;

z = keep\_largest(x + u, K);

% u-update

u = u + (x - z);

end

end

function z = keep\_largest(z, K)

[val pos] = sort(abs(z), 'descend');

z(pos(K+1:end)) = 0;

end

function [L U] = factor(A, rho)

[m, n] = size(A);

if ( m >= n ) % if skinny

L = chol( A'\*A + rho\*speye(n), 'lower' );

else % if fat

L = chol( speye(m) + 1/rho\*(A\*A'), 'lower' );

end

% force matlab to recognize the upper / lower triangular structure

L = sparse(L);

U = sparse(L');

end

Creating the problem and calling the function:

m = 4; % number of examples

n = 3; % number of features

p = 100/n; % sparsity density

% generate sparse solution vector

x = sprandn(n,1,p);

% generate random data matrix

A = randn(m,n);

% normalize columns of A

A = A\*spdiags(1./sqrt(sum(A.^2))', 0, n, n);

% generate measurement b with noise

b = A\*x + sqrt(0.001)\*randn(m,1);

x = regressor\_sel(A, b, p\*n, 1.0)

Sample Output:

x =

0.7423

0.0089

1.3611

### Other Notable Applications of ADMM

1. Some applications of Distributed Reinforcement Learning
2. Filter Denoising
3. Image Restoration
4. Decentralized demand response method in electric vehicle virtual power plant.
5. Used for maximizing Weight Pruning
6. Energy Management of ancillary systems
7. ADMM based privacy-preserving decentralized optimization
8. Approach to informative trajectory planning for multi target tracking

REFERENCES

* 1. Unconstrained and Constrained Optimization Algorithms – Dr Soman K.P
  2. <https://www.youtube.com/watch?v=Xg0ozgCXXB8>
  3. <https://www.youtube.com/watch?v=h8YKBVmRW-E>
  4. <https://web.stanford.edu/~boyd/papers/pdf/admm_slides.pdf>
  5. <https://www.youtube.com/watch?v=h8YKBVmRW-E>
  6. <https://www.youtube.com/watch?v=1tyl_F8j3wA>
  7. <https://scs.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=26607b7a-8a6a-49a6-99f1-a99801302a76>
  8. <https://www.youtube.com/watch?v=05xOrD_-wvQ>
  9. <https://web.stanford.edu/~boyd/papers/admm/>
  10. <https://en.wikipedia.org/wiki/Augmented_Lagrangian_method#Alternating_direction_method_of_multipliers>
  11. <https://arxiv.org/pdf/1705.03412.pdf>
  12. <https://arxiv.org/pdf/1709.05747.pdf>
  13. <https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf>
  14. <https://en.wikipedia.org/wiki/Augmented_Lagrangian_method#General_method>
  15. <http://pwp.gatech.edu/ece-jrom/wp-content/uploads/sites/436/2017/07/16-admm.pdf>
  16. <https://scholar.google.co.in/scholar?start=20&q=notable+applications+of+ADMM&hl=en&as_sdt=0,5&as_vis=1>