

**19 MAT204 MATHEMATICS FOR INTELLIGENT SYSTEMS - III**

**ADMM AND ITS APPLICATIONS**

**GROUP 3**

**TEAM MEMBERS**

Dinesh Kumar M R - CB.EN.U4AIE20011

Rishi Karthigayan S - CB.EN.U4AIE20057

N T Shrish Surya - CB.EN.U4AIE20067

Tsaliki Satya Ganesh Kumar - CB.EN.U4AIE20073

**TABLE OF CONTENTS**

Contents

[**Declaration** 4](#_Toc93410063)

[**Acknowledgment** 5](#_Toc93410064)

[**Abstract** 6](#_Toc93410065)

[**Introduction:** 7](#_Toc93410066)

[**Augmented Lagrangian and the Method of Multipliers:** 8](#_Toc93410067)

[**ADMM and its various forms:** 9](#_Toc93410068)

[**ADMM Form-1:** 9](#_Toc93410069)

[**ADMM Form-2:** 10](#_Toc93410070)

[**Applications of ADMM:** 11](#_Toc93410071)

[**Linear Programming:** 11](#_Toc93410072)

[**MATLAB Implementation :** 12](#_Toc93410073)

[**Example for solving a LP:** 13](#_Toc93410074)

[**Quadratic Programming:** 14](#_Toc93410075)

[**MATLAB Implementation :** 15](#_Toc93410076)

[**Example for solving a QP:** 17](#_Toc93410077)

[**Basis Pursuit:** 17](#_Toc93410078)

[**ADMM formulation:** 18](#_Toc93410079)

[**MATLAB Implementation :** 18](#_Toc93410080)

[**Example problem solved using ADMM:** 20](#_Toc93410081)

[**L1 – Regularized Logistic Regression:** 21](#_Toc93410082)

[**Matlab Implementation:** 21](#_Toc93410083)

[**Example Problem using ADMM:** 24](#_Toc93410084)

[**Support Vector Machine(SVM):** 31](#_Toc93410085)

[**Matlab Implementation:** 31](#_Toc93410086)

[**Example Problem using ADMM:** 33](#_Toc93410087)

[**Intersection of Polyhedra:** 37](#_Toc93410088)

[**Matlab Implementation:** 38](#_Toc93410089)

[**Example Problem using ADMM:** 39](#_Toc93410090)

[**Sparse Inverse Covariance Selection:** 43](#_Toc93410091)

[**Matlab Implementation:** 44](#_Toc93410092)

[**Example Problem using ADMM:** 45](#_Toc93410093)

[**Inference**: 49](#_Toc93410094)

[**Conclusion:** 50](#_Toc93410095)

# **Declaration**

We hereby declare that the project titled “ADMM and its applications'' submitted to the Center for Excellence in Computational Engineering and Networking is a record of the original work done under the guidance of Dr. Neethu Mohan, Assistant Professor in Computational Engineering and Networking, Amrita University.

Signature of the faculty

# **Acknowledgment**

The completion of this project has been made possible by the efforts of many. We would like to thank Dr. Sasangan Ramanathan, Dean - Engineering, and our Mathematics for Intelligent Systems-III faculty, Dr.Neethu Mohan, for their unstinting support. We would also like to extend our gratitude to our friends and family who have aided us throughout the project.

# **Abstract**

Many problems of recent interest in statistics and machine learning can be posed in the framework of convex optimization. Due to the explosion in size and complexity of modern datasets, it is increasingly important to be able to solve problems with a very large number of features or training examples. As a result, both the decentralized collection or storage of these datasets as well as accompanying distributed solution methods are either necessary or at least highly desirable. In this review, we argue that the alternating direction method of multipliers is well suited to distributed convex optimization, and in particular to large-scale problems arising in statistics, machine learning, and related areas. The method was developed in the 1970s, with roots in the 1950s, and is equivalent or closely related to many other algorithms, such as dual decomposition, the method of multipliers, Douglas–Rachford splitting, Spingarn’s method of partial inverses, Dykstra’s alternating projections, Bregman iterative algorithms for l1 problems, proximal methods, and others. After briefly surveying the theory and history of the algorithm, we discuss applications to a wide variety of statistical and machine learning problems of recent interest, including the basis pursuit, regularised logistic regression, sparse inverse covariance selection, support vector machines, and many others. We also discuss linear programming, quadratic programming, and the intersection of polyhedra.

# **Introduction:**

The alternating direction method of multipliers (ADMM) is an algorithm that solves convex optimization problems by breaking them into smaller pieces, each of which is then easier to handle. It has recently found wide application in a number of areas. ADMM Is a simple and powerful iterative algorithm for convex optimization problems. It is much faster than conventional methods. It has emerged as a powerful technique for largescale structured optimization.

ADMM extends the method of multipliers in such a way that we get back some of the decomposability (i.e. ability to parallelize) of standard dual ascent algorithms. It also gives us a flexible framework for incorporating many types of convex constraints. ADMM can be viewed as an attempt to blend the benefits of dual decomposition and augmented Lagrangian methods for constrained optimization.

# **Augmented Lagrangian and the Method of Multipliers:**

Augmented lagrangian method was developed to bring robustness to the dual ascent method. The augmented lagrangian is denoted as . The augmented lagrangian for minimizing the function subjected to is

where >0, it is called penalty parameter and is called a penalty term after modifying this problem is clearly equivalent to the original problem since for any feasible x the term added to the objective is zero.

Now we have to minimize the x value . for that we can apply dual ascent to our modified problem.

At first we have to perform x-minimization and then z-minimization step. Thus they update alternatively

# **ADMM and its various forms:**

## **ADMM Form-1:**

Consider a need to minimize f(x)+g(x),

ADMM works on the principle of increasing the number of variables and solving each of them via multiple iteration. Hence the problem can be re-formulated to,

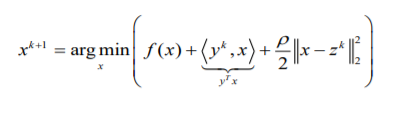
min f(x)+g(z)

Subject to x-z=0

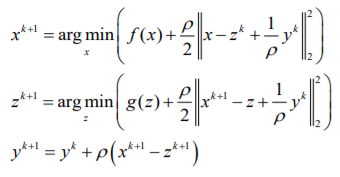
The augmented Lagrangian can be written as :



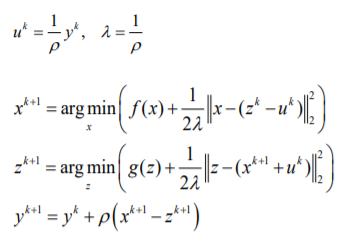
 is the augmented Lagrangian which is used to unconstrained the optimization and decrease the iterations to obtain the solution. While obtaining for nth iteration for a variable , the rest are treated as constants.



In this iteration for x , the constant and other variables are neglected. Hence,



Substituting,



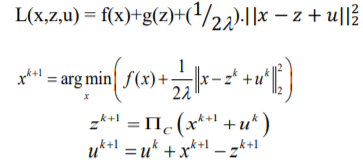
## **ADMM Form-2:**

Consider, min f(x)

Subject to 

This above problem is converted to min f(x) + g(z), Subject to x-z=0 Where g(x) is an indicator function which takes the values 0 if  and infinity otherwise.

The augmented Lagrangian is written as ,





Where represents Projection onto C.

**Shrinkage function in a Nutshell :**

When a threshold is fixed, any input above the threshold will be unaffected and if the input is less than the threshold it will be equated to 0.

# **Applications of ADMM:**

## **Linear Programming:**

In linear programming, the objective is to minimize a closed convex function

; Subject to *Ax=B*

The problem will be converted to the form 2 of ADMM :

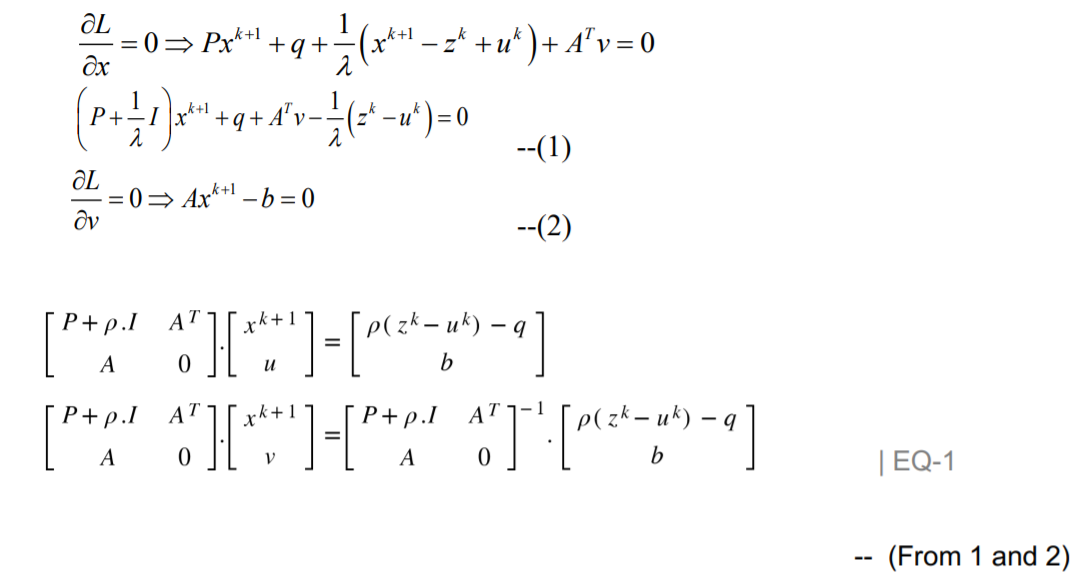
min f(x)+g(z)

Subject to x - z = 0

Where g(z) will be the indicator function.



Partial differentiation w.r.t x,z,u and substituting to 0



where ρ = 1/λ

The very same methodology can be implemented for solving Quadratic equations on x when subject to *Ax=b*. In case of a subject of inequality, the methodology will differ. Usually solving for a linear program, P will be a 0 matrix which reduces f(x) to c’(x)+r;

### **MATLAB Implementation :**

**function [z, history] = linprog(c, A, b, rho, alpha)**

**t\_start = tic;**

**% Global Constants and Defaults**

**QUIET = 0;**

**MAX\_ITER = 1000;**

**ABSTOL = 1e-4;**

**RELTOL = 1e-2;**

**% Data preprocessing**

**[m n] = size(A);**

**% ADMM Solver**

**x = zeros(n,1);**

**z = zeros(n,1);**

**u = zeros(n,1);**

**if ~QUIET**

**fprintf('%3s\t%10s\t%10s\t%10s\t%10s\t%10s\n', 'iter', ...**

**'r norm', 'eps pri', 's norm', 'eps dual', 'objective');**

**end**

**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);**

**% diagnostics, reporting, termination checks**

**history.objval(k) = objective(c, x);**

**history.r\_norm(k) = norm(x - z);**

**history.s\_norm(k) = norm(-rho\*(z - zold));**

**history.eps\_pri(k) = sqrt(n)\*ABSTOL + RELTOL\*max(norm(x), norm(-z));**

**history.eps\_dual(k)= sqrt(n)\*ABSTOL + RELTOL\*norm(rho\*u);**

**if ~QUIET**

**fprintf('%3d\t%10.4f\t%10.4f\t%10.4f\t%10.4f\t%10.2f\n', k, ...**

**history.r\_norm(k), history.eps\_pri(k), ...**

**history.s\_norm(k), history.eps\_dual(k), history.objval(k));**

**end**

**if (history.r\_norm(k) < history.eps\_pri(k) && ...**

**history.s\_norm(k) < history.eps\_dual(k))**

**break;**

**end**

**end**

**if ~QUIET**

**toc(t\_start);**

**end**

**end**

**function obj = objective(c, x)**

**obj = c'\*x;**

**end**

### **Example for solving a LP:**

**randn('state', 0);**

**rand('state', 0);**

**n = 5; % dimension of x**

**m = 4; % number of equality constraints**

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

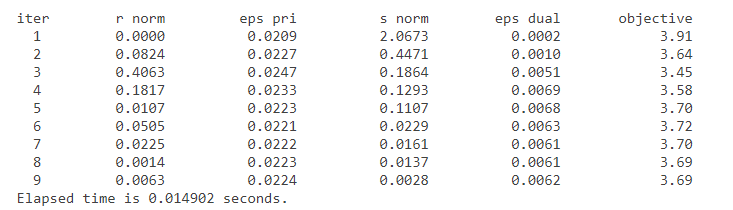
**x0 = abs(randn(n,1)); % create random solution vector**

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

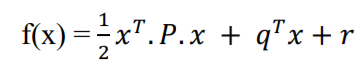
**b = A\*x0;**

**[x history] = linprog(c, A, b, 1.0, 1.0);**

#### **Output:**



## **Quadratic Programming:**

For solving Quadratic equations Subject to lower and upper bound.

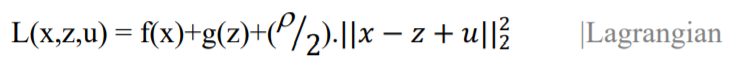
; Subject to : lb<x<ub

The problem will be converted to the form 2 of ADMM :

min f(x)+g(z)

Subject to x - z = 0

where g(z) will be the indicator function.



From EQ-1 : When A,b and v are null –



For increasing efficiency and speed of computation , Consider R to be an upper triangular matrix obtained from Cholesky Factorisation of [P+p.I].

Cholesky Factorisation :

When A is a Hermitian and Positive Deficient Matrix on factorisation we get R, Where A = R\*RT

Hence to increase efficiency ,



### **MATLAB Implementation :**

**function [z, history] = quadprog(P, q, r, lb, ub, rho, alpha)**

**t\_start = tic;**

**%Global constants and defaults**

**QUIET = 0;**

**MAX\_ITER = 1000;**

**ABSTOL = 1e-4;**

**RELTOL = 1e-2;**

**%Data preprocessing**

**n = size(P,1);**

**%ADMM solver**

**x = zeros(n,1);**

**z = zeros(n,1);**

**u = zeros(n,1);**

**if ~QUIET**

**fprintf('%3s\t%10s\t%10s\t%10s\t%10s\t%10s\n', 'iter', ...**

**'r norm', 'eps pri', 's norm', 'eps dual', 'objective');**

**end**

**for k = 1:MAX\_ITER**

**if k > 1**

**x = R \ (R' \ (rho\*(z - u) - q));**

**else**

**R = chol(P + rho\*eye(n));**

**x = R \ (R' \ (rho\*(z - u) - q));**

**end**

**% z-update with relaxation**

**zold = z;**

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

**z = min(ub, max(lb, x\_hat + u));**

**% u-update**

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

**% diagnostics, reporting, termination checks**

**history.objval(k) = objective(P, q, r, x);**

**history.r\_norm(k) = norm(x - z);**

**history.s\_norm(k) = norm(-rho\*(z - zold));**

**history.eps\_pri(k) = sqrt(n)\*ABSTOL + RELTOL\*max(norm(x), norm(-z));**

**history.eps\_dual(k)= sqrt(n)\*ABSTOL + RELTOL\*norm(rho\*u);**

**if ~QUIET**

**fprintf('%3d\t%10.4f\t%10.4f\t%10.4f\t%10.4f\t%10.2f\n', k, ...**

**history.r\_norm(k), history.eps\_pri(k), ...**

**history.s\_norm(k), history.eps\_dual(k), history.objval(k));**

**end**

**if (history.r\_norm(k) < history.eps\_pri(k) && ...**

**history.s\_norm(k) < history.eps\_dual(k))**

**break;**

**end**

**end**

**if ~QUIET**

**toc(t\_start);**

**end**

**end**

**function obj = objective(P, q, r, x)**

**obj = 0.5\*x'\*P\*x + q'\*x + r;**

**end**

### **Example for solving a QP:**

**randn('state', 0);**

**rand('state', 0);**

**n = 100;**

**% generate a well-conditioned positive definite matrix**

**% (for faster convergence)**

**P = rand(n);**

**P = P + P';**

**[V D] = eig(P);**

**P = V\*diag(1+rand(n,1))\*V';**

**q = randn(n,1);**

**r = randn(1);**

**l = randn(n,1);**

**u = randn(n,1);**

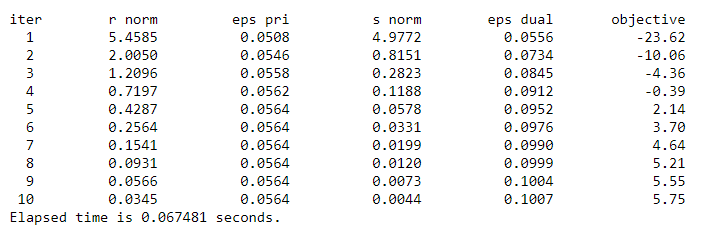
**lb = min(l,u);**

**ub = max(l,u);**

**%Solve problem**

**[x history] = quadprog(P, q, r, lb, ub, 1.0, 1.0);**

#### **Output:**



## **Basis Pursuit:**

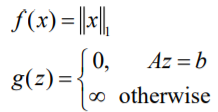
The objective is very simple to minimise the values satisfying *Ax = b*

minimize subject to *Ax = b*

### **ADMM formulation:**

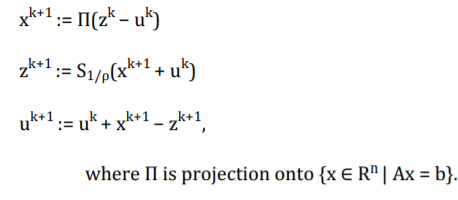
minimize f(x)g(x) subject to x-z vector

where,

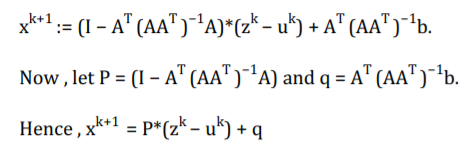


g(z) is like a penalty function when it satisfies the condition Az=b , it will be 0 otherwise it will shoot up to a large value.

The ADMM Algorithm will be :



The x-update, which involves solving a linearly-constrained minimum Euclidean norm problem, can be written explicitly as



### **MATLAB Implementation :**

**function [z, history] = basis\_pursuit(A, b, rho, alpha)**

**t\_start = tic;**

**QUIET = 0;**

**MAX\_ITER = 1000;**

**ABSTOL = 1e-4;**

**RELTOL = 1e-2;**

**[m n] = size(A);**

**x = zeros(n,1);**

**z = zeros(n,1);**

**u = zeros(n,1);**

**if ~QUIET**

**fprintf('%3s\t%10s\t%10s\t%10s\t%10s\t%10s\n', 'iter', ...**

**'r norm', 'eps pri', 's norm', 'eps dual', 'objective');**

**end**

**% precompute static variables for x-update (projection on to Ax=b)**

**AAt = A\*A';**

**P = eye(n) - A' \* (AAt \ A);**

**q = A' \* (AAt \ b);**

**for k = 1:MAX\_ITER**

**% x-update**

**x = P\*(z - u) + q;**

**% z-update with relaxation**

**zold = z;**

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

**z = shrinkage(x\_hat + u, 1/rho);**

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

**% diagnostics, reporting, termination checks**

**history.objval(k) = objective(A, b, x);**

**history.r\_norm(k) = norm(x - z);**

**history.s\_norm(k) = norm(-rho\*(z - zold));**

**history.eps\_pri(k) = sqrt(n)\*ABSTOL + RELTOL\*max(norm(x), norm(-z));**

**history.eps\_dual(k)= sqrt(n)\*ABSTOL + RELTOL\*norm(rho\*u);**

**if ~QUIET**

**fprintf('%3d\t%10.4f\t%10.4f\t%10.4f\t%10.4f\t%10.2f\n', k, ...**

**history.r\_norm(k), history.eps\_pri(k), ...**

**history.s\_norm(k), history.eps\_dual(k), history.objval(k));**

**end**

**if (history.r\_norm(k) < history.eps\_pri(k) && ...**

**history.s\_norm(k) < history.eps\_dual(k))**

**break;**

**end**

**end**

**if ~QUIET**

**toc(t\_start);**

**end**

**end**

**function obj = objective(A, b, x)**

**obj = norm(x,1);**

**end**

**function y = shrinkage(a, kappa)**

**y = max(0, a-kappa) - max(0, -a-kappa);**

**end**

### **Example problem solved using ADMM:**

**rand('seed', 0);**

**randn('seed', 0);**

**n = 3;**

**m = 1;**

**A = randn(m,n);**

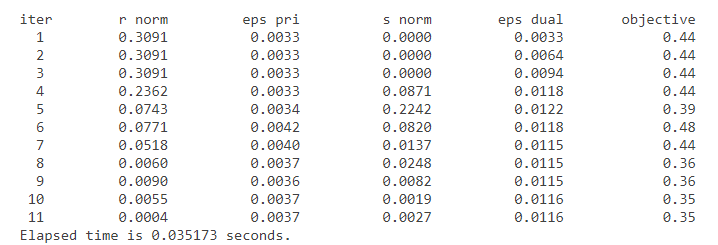
**x = sprandn(n, 1, 0.1\*n);**

**b = A\*x;**

**xtrue = x;**

**[x history] = basis\_pursuit(A, b, 1.0, 1.0);**

#### **Output:**



## **L1 – Regularized Logistic Regression:**

Logistic regression is widely used in machine learning for classification problems. It is well-known that regularization is required to avoid over-fitting, especially when there is only a small number of training examples, or when there are a large number of parameters to be learned. In particular, L1 regularized logistic regression is often used for feature selection and has been shown to have good generalization performance in the presence of many irrelevant features.

L1 regularized logistic regression is a workhorse of machine learning: it is widely used for many classification problems, particularly ones with many features. L1 regularized logistic regression requires solving a convex optimization problem. However, a drawback is that standard algorithms for solving convex optimization problems do not scale well enough to handle the large datasets encountered in many practical settings.

### **Matlab Implementation:**

**function [z, history] = logreg(A, b, mu, rho, alpha)**

**t\_start = tic;**

**Global constants and defaults**

**QUIET = 0;**

**MAX\_ITER = 1000;**

**ABSTOL = 1e-4;**

**RELTOL = 1e-2;**

**Data preprocessing**

**[m, n] = size(A);**

**ADMM solver**

**x = zeros(n+1,1);**

**z = zeros(n+1,1);**

**u = zeros(n+1,1);**

**if ~QUIET**

**fprintf('%3s\t%10s\t%10s\t%10s\t%10s\t%10s\n', 'iter', ...**

**'r norm', 'eps pri', 's norm', 'eps dual', 'objective');**

**end**

**for k = 1:MAX\_ITER**

**% x-update**

**x = update\_x(A, b, u, z, rho);**

**% z-update with relaxation**

**zold = z;**

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

**z = x\_hat + u;**

**z(2:end) = shrinkage(z(2:end), (m\*mu)/rho);**

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

**% diagnostics, reporting, termination checks**

**history.objval(k) = objective(A, b, mu, x, z);**

**history.r\_norm(k) = norm(x - z);**

**history.s\_norm(k) = norm(rho\*(z - zold));**

**history.eps\_pri(k) = sqrt(n)\*ABSTOL + RELTOL\*max(norm(x), norm(z));**

**history.eps\_dual(k)= sqrt(n)\*ABSTOL + RELTOL\*norm(rho\*u);**

**if ~QUIET**

**fprintf('%3d\t%10.4f\t%10.4f\t%10.4f\t%10.4f\t%10.2f\n', k, ...**

**history.r\_norm(k), history.eps\_pri(k), ...**

**history.s\_norm(k), history.eps\_dual(k), history.objval(k));**

**end**

**if history.r\_norm(k) < history.eps\_pri(k) && ...**

**history.s\_norm(k) < history.eps\_dual(k)**

**break;**

**end**

**end**

**if ~QUIET**

**toc(t\_start);**

**end**

**end**

**function obj = objective(A, b, mu, x, z)**

**m = size(A,1);**

**obj = sum(log(1 + exp(-A\*x(2:end) - b\*x(1)))) + m\*mu\*norm(z,1);**

**end**

**function x = update\_x(A, b, u, z, rho, x0)**

**% solve the x update**

**% minimize [ -logistic(x\_i) + (rho/2)||x\_i - z^k + u^k||^2 ]**

**% via Newton's method; for a single subsystem only.**

**alpha = 0.1;**

**BETA = 0.5;**

**TOLERANCE = 1e-5;**

**MAX\_ITER = 50;**

**[m n] = size(A);**

**I = eye(n+1);**

**if exist('x0', 'var')**

**x = x0;**

**else**

**x = zeros(n+1,1);**

**end**

**C = [-b -A];**

**f = @(w) (sum(log(1 + exp(C\*w))) + (rho/2)\*norm(w - z + u).^2);**

**for iter = 1:MAX\_ITER**

**fx = f(x);**

**g = C'\*(exp(C\*x)./(1 + exp(C\*x))) + rho\*(x - z + u);**

**H = C' \* diag(exp(C\*x)./(1 + exp(C\*x)).^2) \* C + rho\*I;**

**dx = -H\g; % Newton step**

**dfx = g'\*dx; % Newton decrement**

**if abs(dfx) < TOLERANCE**

**break;**

**end**

**% backtracking**

**t = 1;**

**while f(x + t\*dx) > fx + alpha\*t\*dfx**

**t = BETA\*t;**

**end**

**x = x + t\*dx;**

**end**

**end**

**function z = shrinkage(a, kappa)**

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

**end**

### **Example Problem using ADMM:**

**Generate problem data**

**rand('seed', 0);**

**randn('seed', 0);**

**n = 5;**

**m = 20;**

**w = sprandn(n, 1, 0.1); % N(0,1), 10% sparse**

**v = randn(1); % random intercept**

**X = sprandn(m, n, 10/n);**

**btrue = sign(X\*w + v);**

**% noise is function of problem size use 0.1 for large problem**

**b = sign(X\*w + v + sqrt(0.1)\*randn(m,1)); % labels with noise**

**A = spdiags(b, 0, m, m) \* X;**

**ratio = sum(b == 1)/(m);**

**mu = 0.1 \* 1/m \* norm((1-ratio)\*sum(A(b==1,:),1) + ratio\*sum(A(b==-1,:),1), 'inf');**

**x\_true = [v; w];**

**Solve problem**

**[x history] = logreg(A, b, mu, 1.0, 1.0);**

**iter r norm eps pri s norm eps dual objective**

**1 0.3774 0.0199 1.8188 0.0040 5.56**

**2 0.0416 0.0276 0.9767 0.0042 4.78**

**3 0.0202 0.0340 0.6855 0.0041 4.32**

**4 0.0472 0.0390 0.5355 0.0038 4.06**

**5 0.0515 0.0431 0.4411 0.0036 3.89**

**6 0.0486 0.0466 0.3754 0.0034 3.78**

**7 0.0435 0.0497 0.3265 0.0033 3.70**

**8 0.0380 0.0524 0.2886 0.0033 3.64**

**9 0.0328 0.0549 0.2581 0.0033 3.60**

**10 0.0283 0.0571 0.2331 0.0033 3.57**

**11 0.0565 0.0591 0.2047 0.0030 3.55**

**12 0.0262 0.0609 0.1815 0.0029 3.54**

**13 0.0142 0.0625 0.1636 0.0029 3.53**

**14 0.0106 0.0640 0.1487 0.0029 3.53**

**15 0.0089 0.0653 0.1360 0.0028 3.52**

**16 0.0076 0.0666 0.1250 0.0028 3.52**

**17 0.0066 0.0677 0.1152 0.0028 3.52**

**18 0.0058 0.0688 0.1067 0.0028 3.52**

**19 0.0051 0.0698 0.0990 0.0028 3.53**

**20 0.0046 0.0707 0.0921 0.0028 3.53**

**21 0.0041 0.0715 0.0859 0.0028 3.53**

**22 0.0037 0.0723 0.0803 0.0028 3.53**

**23 0.0033 0.0731 0.0753 0.0028 3.54**

**24 0.0030 0.0738 0.0706 0.0028 3.54**

**25 0.0027 0.0744 0.0664 0.0028 3.55**

**26 0.0025 0.0751 0.0625 0.0028 3.55**

**27 0.0023 0.0756 0.0589 0.0028 3.55**

**28 0.0021 0.0762 0.0556 0.0028 3.56**

**29 0.0019 0.0767 0.0525 0.0028 3.56**

**30 0.0018 0.0772 0.0496 0.0028 3.56**

**31 0.0016 0.0777 0.0470 0.0028 3.57**

**32 0.0015 0.0781 0.0445 0.0028 3.57**

**33 0.0014 0.0785 0.0422 0.0028 3.58**

**34 0.0013 0.0789 0.0401 0.0028 3.58**

**35 0.0012 0.0793 0.0381 0.0028 3.58**

**36 0.0011 0.0797 0.0362 0.0028 3.59**

**37 0.0011 0.0800 0.0344 0.0028 3.59**

**38 0.0010 0.0804 0.0327 0.0028 3.59**

**39 0.0009 0.0807 0.0312 0.0028 3.59**

**40 0.0009 0.0810 0.0297 0.0028 3.60**

**41 0.0008 0.0812 0.0283 0.0028 3.60**

**42 0.0008 0.0815 0.0270 0.0028 3.60**

**43 0.0007 0.0818 0.0257 0.0028 3.61**

**44 0.0007 0.0820 0.0246 0.0028 3.61**

**45 0.0006 0.0823 0.0234 0.0028 3.61**

**46 0.0006 0.0825 0.0224 0.0028 3.61**

**47 0.0006 0.0827 0.0214 0.0028 3.61**

**48 0.0005 0.0829 0.0204 0.0028 3.62**

**49 0.0005 0.0831 0.0196 0.0028 3.62**

**50 0.0005 0.0833 0.0187 0.0028 3.62**

**51 0.0005 0.0835 0.0179 0.0028 3.62**

**52 0.0004 0.0836 0.0171 0.0028 3.62**

**53 0.0004 0.0838 0.0164 0.0028 3.63**

**54 0.0004 0.0839 0.0157 0.0028 3.63**

**55 0.0004 0.0841 0.0150 0.0028 3.63**

**56 0.0004 0.0842 0.0144 0.0028 3.63**

**57 0.0003 0.0844 0.0138 0.0028 3.63**

**58 0.0003 0.0845 0.0132 0.0028 3.63**

**59 0.0003 0.0846 0.0126 0.0028 3.64**

**60 0.0003 0.0848 0.0121 0.0028 3.64**

**61 0.0003 0.0849 0.0116 0.0028 3.64**

**62 0.0003 0.0850 0.0111 0.0028 3.64**

**63 0.0003 0.0851 0.0107 0.0028 3.64**

**64 0.0002 0.0852 0.0102 0.0028 3.64**

**65 0.0002 0.0853 0.0098 0.0028 3.64**

**66 0.0002 0.0854 0.0094 0.0028 3.64**

**67 0.0002 0.0855 0.0090 0.0028 3.65**

**68 0.0002 0.0856 0.0087 0.0028 3.65**

**69 0.0002 0.0856 0.0083 0.0028 3.65**

**70 0.0002 0.0857 0.0080 0.0028 3.65**

**71 0.0002 0.0858 0.0077 0.0028 3.65**

**72 0.0002 0.0859 0.0074 0.0028 3.65**

**73 0.0002 0.0859 0.0071 0.0028 3.65**

**74 0.0002 0.0860 0.0068 0.0028 3.65**

**75 0.0001 0.0861 0.0065 0.0028 3.65**

**76 0.0001 0.0861 0.0063 0.0028 3.65**

**77 0.0001 0.0862 0.0060 0.0028 3.65**

**78 0.0001 0.0863 0.0058 0.0028 3.66**

**79 0.0001 0.0863 0.0056 0.0028 3.66**

**80 0.0001 0.0864 0.0053 0.0028 3.66**

**81 0.0001 0.0864 0.0051 0.0028 3.66**

**82 0.0001 0.0865 0.0049 0.0028 3.66**

**83 0.0001 0.0865 0.0047 0.0028 3.66**

**84 0.0001 0.0866 0.0046 0.0028 3.66**

**85 0.0001 0.0866 0.0044 0.0028 3.66**

**86 0.0001 0.0866 0.0042 0.0028 3.66**

**87 0.0001 0.0867 0.0040 0.0028 3.66**

**88 0.0001 0.0867 0.0039 0.0028 3.66**

**89 0.0001 0.0868 0.0037 0.0028 3.66**

**90 0.0001 0.0868 0.0036 0.0028 3.66**

**91 0.0001 0.0868 0.0034 0.0028 3.66**

**92 0.0001 0.0869 0.0033 0.0028 3.66**

**93 0.0001 0.0869 0.0032 0.0028 3.66**

**94 0.0001 0.0869 0.0031 0.0028 3.66**

**95 0.0001 0.0870 0.0029 0.0028 3.66**

**96 0.0001 0.0870 0.0028 0.0028 3.66**

**97 0.0001 0.0870 0.0027 0.0028 3.66**

**Elapsed time is 0.258555 seconds.**

**Reporting**

**K = length(history.objval);**

**h = figure;**

**plot(1:K, history.objval, 'k', 'MarkerSize', 10, 'LineWidth', 2);**

**ylabel('f(x^k) + g(z^k)'); xlabel('iter (k)');**

**g = figure;**

**subplot(2,1,1);**

**semilogy(1:K, max(1e-8, history.r\_norm), 'k', ...**

**1:K, history.eps\_pri, 'k--', 'LineWidth', 2);**

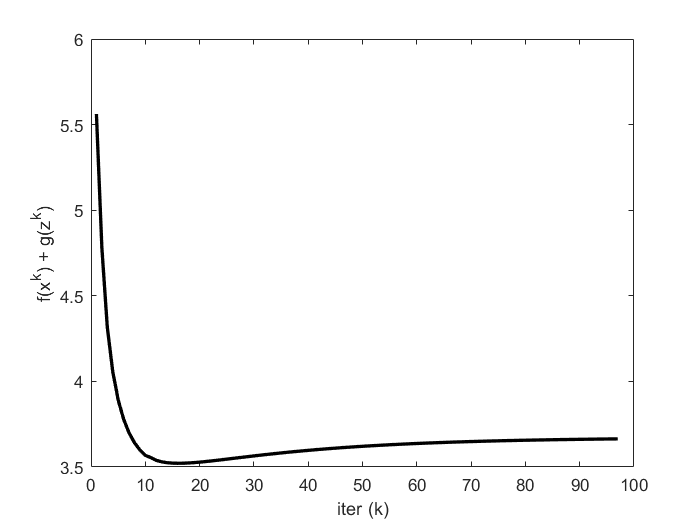
**ylabel('||r||\_2');**

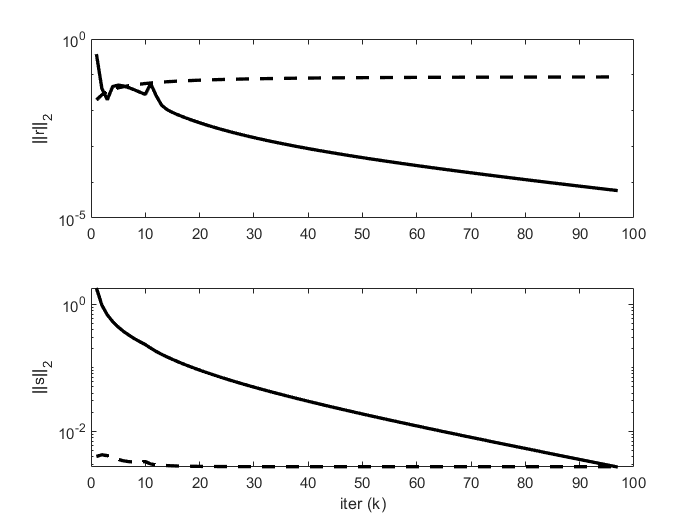
**subplot(2,1,2);**

**semilogy(1:K, max(1e-8, history.s\_norm), 'k', ...**

**1:K, history.eps\_dual, 'k--', 'LineWidth', 2);**

**ylabel('||s||\_2'); xlabel('iter (k)');**

****

****

## **Support Vector Machine(SVM):**

Support Vector Machine(SVM) is a supervised machine learning algorithm used for both classification and regression. Though we say regression problems it is best suited for classification also. The objective of the SVM algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points. The dimension of the hyperplane depends upon the number of features. If the number of input features is two, then the hyperplane is just a line. If the number of input features is three, then the hyperplane becomes a 2-D plane. It becomes difficult when the number of features exceeds three.

### **Matlab Implementation:**

**function [xave, history] = linear\_svm(A, lambda, p, rho, alpha)**

**t\_start = tic;**

**Global constants and defaults**

**QUIET = 0;**

**MAX\_ITER = 1000;**

**ABSTOL = 1e-4;**

**RELTOL = 1e-2;**

**Data preprocessing**

**[m, n] = size(A);**

**N = max(p);**

**% group samples together**

**for i = 1:N,**

**tmp{i} = A(p==i,:);**

**end**

**A = tmp;**

**ADMM solver**

**x = zeros(n,N);**

**z = zeros(n,N);**

**u = zeros(n,N);**

**if ~QUIET**

**fprintf('%3s\t%10s\t%10s\t%10s\t%10s\t%10s\n', 'iter', ...**

**'r norm', 'eps pri', 's norm', 'eps dual', 'objective');**

**end**

**for k = 1:MAX\_ITER**

**% x-update**

**for i = 1:N,**

**cvx\_begin quiet**

**variable x\_var(n)**

**minimize ( sum(pos(A{i}\*x\_var + 1)) + rho/2\*sum\_square(x\_var - z(:,i) + u(:,i)) )**

**cvx\_end**

**x(:,i) = x\_var;**

**end**

**xave = mean(x,2);**

**% z-update with relaxation**

**zold = z;**

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

**z = N\*rho/(1/lambda + N\*rho)\*mean( x\_hat + u, 2 );**

**z = z\*ones(1,N);**

**% u-update**

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

**% diagnostics, reporting, termination checks**

**history.objval(k) = objective(A, lambda, p, x, z);**

**history.r\_norm(k) = norm(x - z);**

**history.s\_norm(k) = norm(-rho\*(z - zold));**

**history.eps\_pri(k) = sqrt(n)\*ABSTOL + RELTOL\*max(norm(x), norm(-z));**

**history.eps\_dual(k)= sqrt(n)\*ABSTOL + RELTOL\*norm(rho\*u);**

**if ~QUIET**

**fprintf('%3d\t%10.4f\t%10.4f\t%10.4f\t%10.4f\t%10.2f\n', k, ...**

**history.r\_norm(k), history.eps\_pri(k), ...**

**history.s\_norm(k), history.eps\_dual(k), history.objval(k));**

**end**

**if (history.r\_norm(k) < history.eps\_pri(k) && ...**

**history.s\_norm(k) < history.eps\_dual(k))**

**break;**

**end**

**end**

**if ~QUIET**

**toc(t\_start);**

**end**

**end**

**function obj = objective(A, lambda, p, x, z)**

**obj = hinge\_loss(A,x) + 1/(2\*lambda)\*sum\_square(z(:,1));**

**end**

**function val = hinge\_loss(A,x)**

**val = 0;**

**for i = 1:length(A)**

**val = val + sum(pos(A{i}\*x(:,i) + 1));**

**end**

**end**

### **Example Problem using ADMM:**

**rand('seed', 0);**

**randn('seed', 0);**

**n = 2;**

**m = 200;**

**N = m/2;**

**M = m/2;**

**% positive examples**

**Y = [1.5+0.9\*randn(1,0.6\*N), 1.5+0.7\*randn(1,0.4\*N);**

**2\*(randn(1,0.6\*N)+1), 2\*(randn(1,0.4\*N)-1)];**

**% negative examples**

**X = [-1.5+0.9\*randn(1,0.6\*M), -1.5+0.7\*randn(1,0.4\*M);**

**2\*(randn(1,0.6\*M)-1), 2\*(randn(1,0.4\*M)+1)];**

**x = [X Y];**

**y = [ones(1,N) -ones(1,M)];**

**A = [ -((ones(n,1)\*y).\*x)' -y'];**

**xdat = x';**

**lambda = 1.0;**

**% partition the examples up in the worst possible way**

**% (subsystems only have positive or negative examples)**

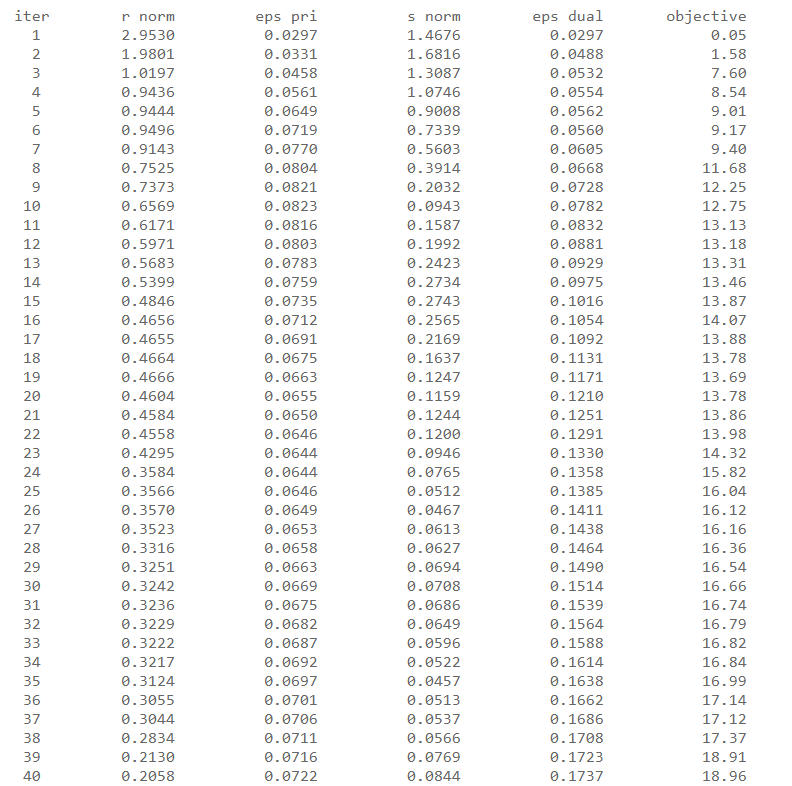
**p = zeros(1,m);**

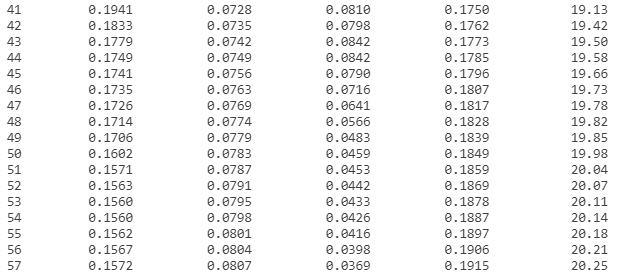
**p(y == 1) = sort(randi([1 10], sum(y==1),1));**

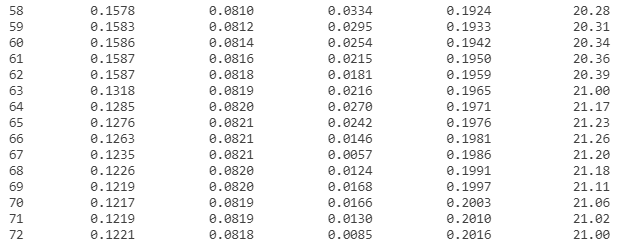
**p(y == -1) = sort(randi([11 20], sum(y==-1),1));**

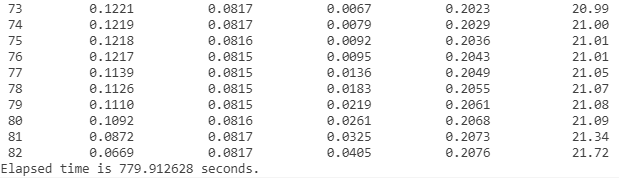
**Solve problem**

**[x history] = linear\_svm(A, lambda, p, 1.0, 1.0);**

****

****

****

****

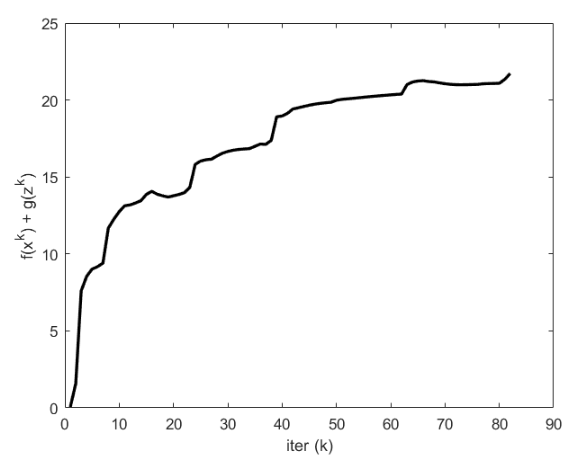
**Reporting**

**K = length(history.objval);**

**h = figure;**

**plot(1:K, history.objval, 'k', 'MarkerSize', 10, 'LineWidth', 2);**

**ylabel('f(x^k) + g(z^k)'); xlabel('iter (k)');**

****

**g = figure;**

**subplot(2,1,1);**

**semilogy(1:K, max(1e-8, history.r\_norm), 'k', ...**

**1:K, history.eps\_pri, 'k--', 'LineWidth', 2);**

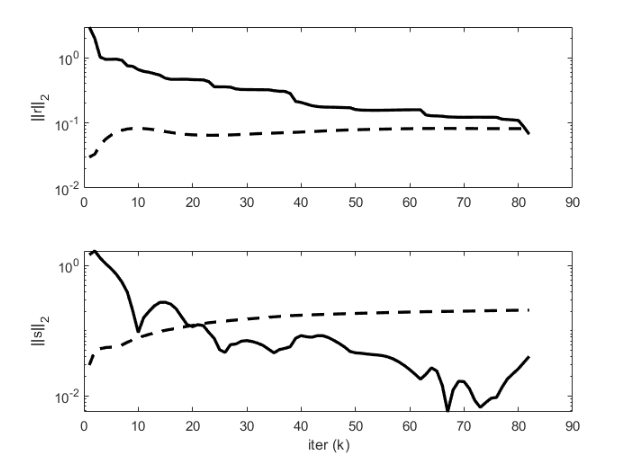
**ylabel('||r||\_2');**

**subplot(2,1,2);**

**semilogy(1:K, max(1e-8, history.s\_norm), 'k', ...**

**1:K, history.eps\_dual, 'k--', 'LineWidth', 2);**

**ylabel('||s||\_2'); xlabel('iter (k)');**

****

## **Intersection of Polyhedra:**

The intersection detection problem is a subset of collision detection for more complex forms, which plays a major role in robotics, computer animation, and mechanical simulation. Intersection detection is also used in computer graphics as a component of accelerated data structures such as bounding volume hierarchies and Kd-trees. In the latter scenario, an item of interest (such as a ray, a view frustum, or another hierarchy) is compared to the geometric forms that bind each node in the hierarchy to see if they overlap. Simple forms (boxes, spheres) for which quick intersection detection algorithms exist are used as bounding shapes.

### **Matlab Implementation:**

**function [z, history] = polyhedra\_intersection(A1, b1, A2, b2, rho, alpha)**

**t\_start = tic;**

**Global constants and defaults**

**QUIET = 0;**

**MAX\_ITER = 1000;**

**ABSTOL = 1e-4;**

**RELTOL = 1e-2;**

**n = size(A1,2);**

## 

**ADMM solver**

**x = zeros(n,1);**

**z = zeros(n,1);**

**u = zeros(n,1);**

**if ~QUIET**

**fprintf('%3s\t%10s\t%10s\t%10s\t%10s\t%10s\n', 'iter', ...**

**'r norm', 'eps pri', 's norm', 'eps dual', 'objective');**

**end**

**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);**

**% diagnostics, reporting, termination checks**

**history.objval(k) = 0;**

**history.r\_norm(k) = norm(x - z);**

**history.s\_norm(k) = norm(-rho\*(z - zold));**

**history.eps\_pri(k) = sqrt(n)\*ABSTOL + RELTOL\*max(norm(x), norm(-z));**

**history.eps\_dual(k)= sqrt(n)\*ABSTOL + RELTOL\*norm(rho\*u);**

**if ~QUIET**

**fprintf('%3d\t%10.4f\t%10.4f\t%10.4f\t%10.4f\t%10.2f\n', k, ...**

**history.r\_norm(k), history.eps\_pri(k), ...**

**history.s\_norm(k), history.eps\_dual(k), history.objval(k));**

**end**

**if (history.r\_norm(k) < history.eps\_pri(k) && ...**

**history.s\_norm(k) < history.eps\_dual(k))**

**break;**

**end**

**end**

**if ~QUIET**

**toc(t\_start);**

**end**

**end**

### **Example Problem using ADMM:**

**randn('state', 0);**

**rand('state', 0);**

**n = 5; % dimension of variable**

**m1 = 16; % number of faces for polyhedra 1**

**m2 = 18; % number of faces for polyhedra 2**

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

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

**% consider the following picture:**

**%**

**% a1**

**% c ---------> x**

**%**

**% from the center "c", we travel along vector "a1" (not necessarily a unit**

**% vector) until we reach x. at "x", a1'x = b. a point y is to the left of x**

**% if a1'y <= b.**

**%**

**% pick m1 random directions with different magnitudes**

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

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

**% vectors in A1 and taking its inner product with A1.**

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

**% pick m2 random directions with different magnitudes**

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

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

**% vectors in A1 and taking its inner product with A1.**

**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**

**cvx\_begin quiet**

**variables x(n) y(n)**

**minimize sum\_square(x - y)**

**subject to**

**A1\*x <= b1**

**A2\*y <= b2**

**cvx\_end**

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

**% distance**

**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**

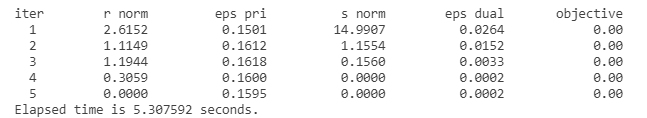
**b1 = diag(A1\*(c1\*ones(1,m1) + A1'));**

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

**end**

**Solve problem**

**[x history] = polyhedra\_intersection(A1, b1, A2, b2, 1.0, 1.0);**

****

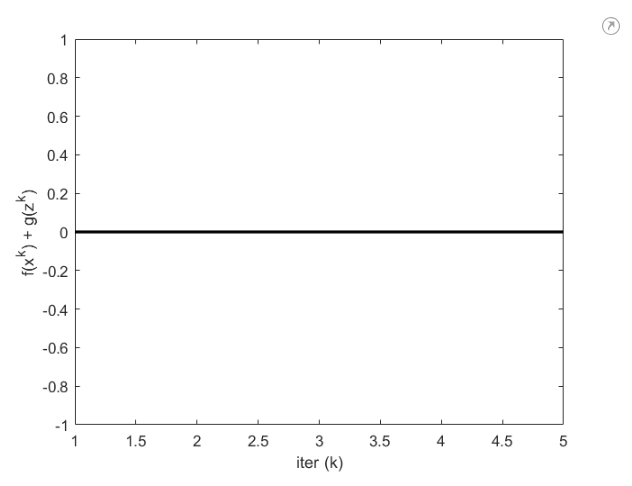
**Reporting**

**K = length(history.objval);**

**h = figure;**

**plot(1:K, history.objval, 'k', 'MarkerSize', 10, 'LineWidth', 2);**

**ylabel('f(x^k) + g(z^k)'); xlabel('iter (k)');**

****

**g = figure;**

**subplot(2,1,1);**

**semilogy(1:K, max(1e-8, history.r\_norm), 'k', ...**

**1:K, history.eps\_pri, 'k--', 'LineWidth', 2);**

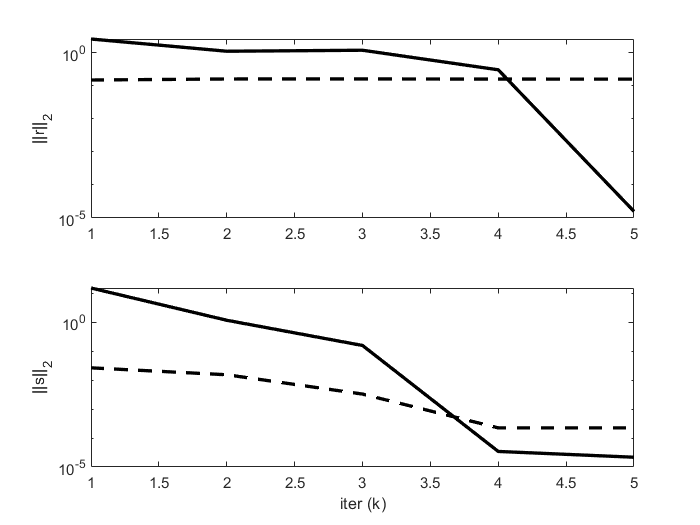
**ylabel('||r||\_2');**

**subplot(2,1,2);**

**semilogy(1:K, max(1e-8, history.s\_norm), 'k', ...**

**1:K, history.eps\_dual, 'k--', 'LineWidth', 2);**

**ylabel('||s||\_2'); xlabel('iter (k)');**

****

**Compare to alternating projections**

**% Compare to alternating projections**

**MAX\_ITER = 10;**

**x = zeros(n,1);**

**z = 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))**

**subject to**

**A1\*x <= b1**

**cvx\_end**

**% z-update with relaxation**

**zold = z;**

**% use cvx to find point in second polyhedra**

**cvx\_begin quiet**

**variable z(n)**

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

**subject to**

**A2\*z <= b2**

**cvx\_end**

**history1.r\_norm(k) = norm(x - z);**

**history1.s\_norm(k) = norm((z - zold));**

**end**

**g = figure;**

**subplot(2,1,1);**

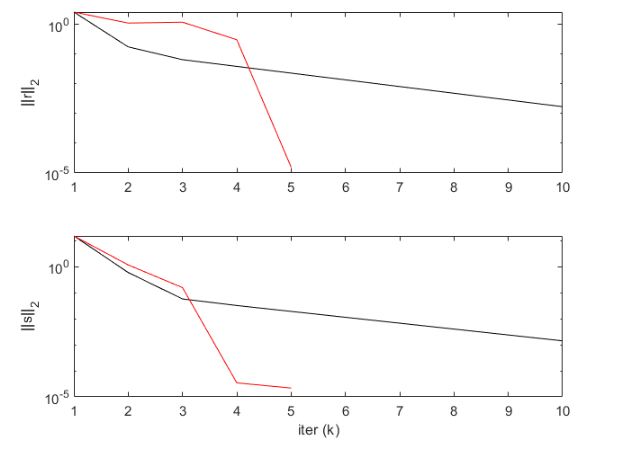
**semilogy(1:MAX\_ITER, max(1e-8, history1.r\_norm), 'k', 1:K, max(1e-8, history.r\_norm), 'r');**

**ylabel('||r||\_2');**

**subplot(2,1,2);**

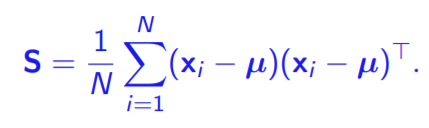
**semilogy(1:MAX\_ITER, max(1e-8, history1.s\_norm), 'k', 1:K, max(1e-8, history.s\_norm), 'r');**

**ylabel('||s||\_2'); xlabel('iter (k)');**

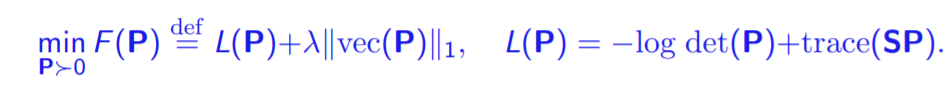
****

## **Sparse Inverse Covariance Selection:**

It is a sequence of positive definite inverse covariance matrices that converge to a sparse matrix, while the sequence of Y k's is a sequence of sparse matrices that converges to a positive definite inverse covariance matrix. To estimate a probabilistic model (e.g. a Gaussian model), estimating the precision matrix, that is the inverse covariance matrix, is as important as estimating the covariance matrix. Indeed a Gaussian model is parametrized by the precision matrix. Given an empirical covariance matrix S,



we find a sparse inverse covariance matrix P to represent the data.



where L is the negative log-likelihood function and the term is a

sparsity inducing regularizer.

### **Matlab Implementation:**

**function [Z, history] = covsel(D, lambda, rho, alpha)**

**t\_start = tic;**

**Global constants and defaults**

**QUIET = 0;**

**MAX\_ITER = 1000;**

**ABSTOL = 1e-4;**

**RELTOL = 1e-2;**

**Data preprocessing**

**S = cov(D);**

**n = size(S,1);**

**ADMM solver**

**X = zeros(n);**

**Z = zeros(n);**

**U = zeros(n);**

**if ~QUIET**

**fprintf('%3s\t%10s\t%10s\t%10s\t%10s\t%10s\n', 'iter', ...**

**'r norm', 'eps pri', 's norm', 'eps dual', 'objective');**

**end**

**for k = 1:MAX\_ITER**

**% x-update**

**[Q,L] = eig(rho\*(Z - U) - S);**

**es = diag(L);**

**xi = (es + sqrt(es.^2 + 4\*rho))./(2\*rho);**

**X = Q\*diag(xi)\*Q';**

**% z-update with relaxation**

**Zold = Z;**

**X\_hat = alpha\*X + (1 - alpha)\*Zold;**

**Z = shrinkage(X\_hat + U, lambda/rho);**

**U = U + (X\_hat - Z);**

**% diagnostics, reporting, termination checks**

**history.objval(k) = objective(S, X, Z, lambda);**

**history.r\_norm(k) = norm(X - Z, 'fro');**

**history.s\_norm(k) = norm(-rho\*(Z - Zold),'fro');**

**history.eps\_pri(k) = sqrt(n\*n)\*ABSTOL + RELTOL\*max(norm(X,'fro'), norm(Z,'fro'));**

**history.eps\_dual(k)= sqrt(n\*n)\*ABSTOL + RELTOL\*norm(rho\*U,'fro');**

**if ~QUIET**

**fprintf('%3d\t%10.4f\t%10.4f\t%10.4f\t%10.4f\t%10.2f\n', k, ...**

**history.r\_norm(k), history.eps\_pri(k), ...**

**history.s\_norm(k), history.eps\_dual(k), history.objval(k));**

**end**

**if (history.r\_norm(k) < history.eps\_pri(k) && ...**

**history.s\_norm(k) < history.eps\_dual(k))**

**break;**

**end**

**end**

**if ~QUIET**

**toc(t\_start);**

**end**

**end**

**function obj = objective(S, X, Z, lambda)**

**obj = trace(S\*X) - log(det(X)) + lambda\*norm(Z(:), 1);**

**end**

**function y = shrinkage(a, kappa)**

**y = max(0, a-kappa) - max(0, -a-kappa);**

**end**

### **Example Problem using ADMM:**

**randn('seed', 0);**

**rand('seed', 0);**

**n = 100; % number of features**

**N = 10\*n; % number of samples**

**% generate a sparse positive definite inverse covariance matrix**

**Sinv = diag(abs(ones(n,1)));**

**idx = randsample(n^2, 0.001\*n^2);**

**Sinv(idx) = ones(numel(idx), 1);**

**Sinv = Sinv + Sinv'; % make symmetric**

**if min(eig(Sinv)) < 0 % make positive definite**

**Sinv = Sinv + 1.1\*abs(min(eig(Sinv)))\*eye(n);**

**end**

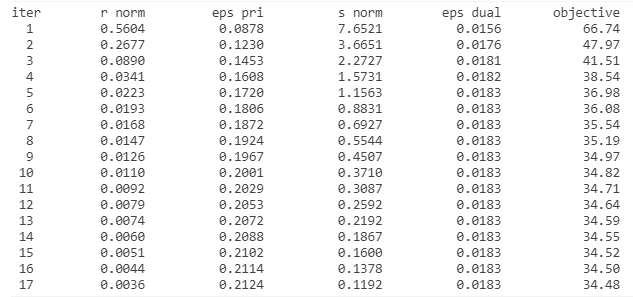
**S = inv(Sinv);**

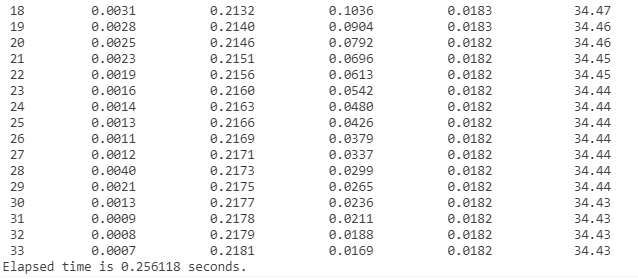
**% generate Gaussian samples**

**D = mvnrnd(zeros(1,n), S, N);**

**Solve problem**

**[X, history] = covsel(D, 0.01, 1, 1);**

****

****

**Reporting**

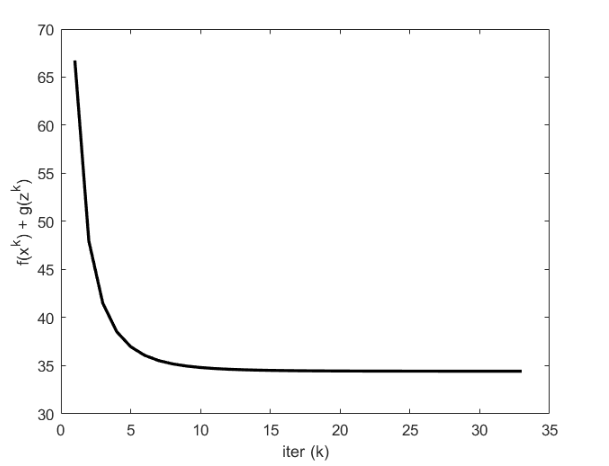
**K = length(history.objval);**

**X\_admm = X;**

**h = figure;**

**plot(1:K, history.objval, 'k', 'MarkerSize', 10, 'LineWidth', 2);**

**ylabel('f(x^k) + g(z^k)'); xlabel('iter (k)');**

****

**g = figure;**

**subplot(2,1,1);**

**semilogy(1:K, max(1e-8, history.r\_norm), 'k', ...**

**1:K, history.eps\_pri, 'k--', 'LineWidth', 2);**

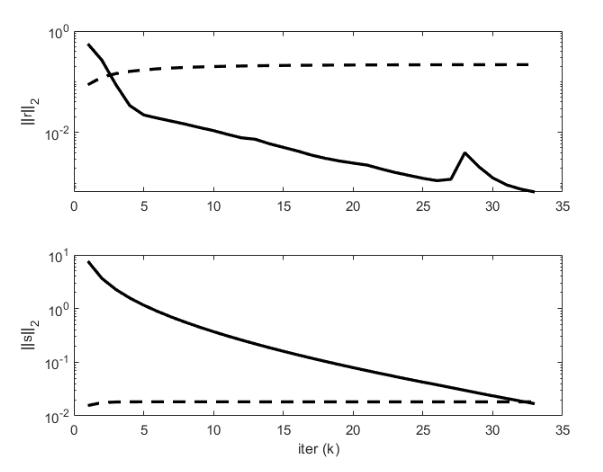
**ylabel('||r||\_2');**

**subplot(2,1,2);**

**semilogy(1:K, max(1e-8, history.s\_norm), 'k', ...**

**1:K, history.eps\_dual, 'k--', 'LineWidth', 2);**

**ylabel('||s||\_2'); xlabel('iter (k)');**

****

# **Inference**:

* ADMM has the benefit of being extremely simple to implement, and it maps onto several standard distributed programming models reasonably well.​
* ADMM was developed over a generation ago, with its roots stretching far in advance of the Internet, distributed and cloud computing systems, massive high-dimensional datasets, and the associated large scale applied statistical problems.​
* ​Despite this, it appears to be well suited to the modern regime and has the important benefit of being quite general in its scope and applicability.​
* ADMM builds on existing algorithms for single machines, and so can be viewed as a modular coordination algorithm that ‘incentivizes’ a set of simpler algorithms to collaborate to solve much larger global problems together than they could on their own.**​**

# **Conclusion:**