Topics on High-Dimensional Data Analytics

Optimization Methods II

Kamran Paynabar, Ph.D.
Associate Professor
School of Industrial and Systems Engineering

Proximal Gradient Descent

Learning Objectives

- Define Decomposable functions
- Use Proximal gradient descent
- Accelerate Proximal gradient descent

Decomposable Functions

Consider a convex function f that can be decomposed into two functions g and h in the following form:

$$f(x) = g(x) + h(x)$$

Suppose, we want to find

$$x^* = argmin\, f(x) \quad for\, x \in R^n$$

If both g and h are **convex and differentiable** then we could use gradient descent algorithm by minimizing the quadratic approximation of *f*.

$$f(z) = f(x) + \nabla f(x)^T(z-x) + \frac{1}{2t}||z-x||_2^2$$

Which results in

$$x_{k+1} = x_k - t\nabla f(x_k)$$

Decomposable Functions

Consider a convex function f that can be decomposed into two functions g and h in the following form:

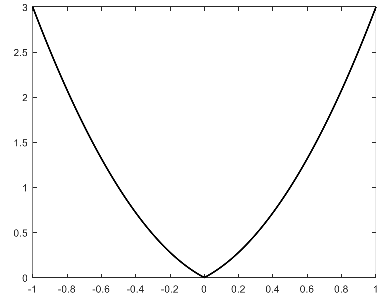
$$f(x) = g(x) + h(x)$$

Suppose, we want to find

$$x^* = argmin\, f(x) \quad for\, x \in R^n$$

Now assume, g is **convex** and **differentiable**,
and h is **convex** (*may or may not be differentiable*)

Example: $f(x) = x^T A x + |x|_1$
(in real numbers, $f(x) = 2x^2 + |x|$)



Proximal Gradient Descent

- How can we efficiently solve: $x^* = argmin(g(x) + h(x)) \quad for\ x \in R^d$ using an iterative method?

- Because g is a differentiable function, we can approximate it around point x using quadratic approximation as following:

$$g(z) = g(x) + \nabla g(x)^T(z-x) + \frac{1}{2t}\left|\left|z-x\right|\right|_2^2$$

- Therefore, the function *f* can be rewritten as

$$\begin{aligned} x^+ &= \arg\min_z \left(g(x) + \nabla g(x)^T (z-x) + \frac{1}{2t}||z-x||_2^2 + h(z) \right) \\ &= \arg\min_z \frac{1}{2t}||z-(x-t\nabla g(x))||_2^2 + h(z) \end{aligned}$$

Proximal Gradient Descent

- Let us define the proximal function as $prox_t(x) = \arg\min_z \left(\frac{1}{2t}||z-x||_2^2 + h(z) \right)$
- Using the proximal function, the optimization problem that determines the direction in the next iteration can be written by

$$prox_t(x - t\nabla g(x)) = \arg\min \frac{1}{2t}\|z-(x-t\nabla g(x))\|_2^2 + h(z)$$

- The **proximal gradient descent algorithm** gives the next value as following:

$$x_k = prox_{t_k}(x_{k-1} - t_k \nabla g(x_{k-1}))$$

```
Algorithm 1 Proximal Gradient Descent
Define prox_t(x)
Initialize x_0
for k = 1 : K do
    Compute x_k = prox_t(x_{k-1} - t∇g(x_{k-1}))
end for
```

- Note that proximal gradient descent works well if the $prox$ function can be computed easily.

Proximal Gradient Descent - Example

Consider the objective of function of LASSO regression

$$f(x) = \underbrace{\frac{1}{2}||y - Ax||_2^2}_{g(x)} + \underbrace{\lambda||x||_1}_{h(x)}$$

Then

$$prox_t(a) = \arg\min_z \frac{1}{2t}\|z - a\|_2^2 + \lambda||z||_1 = S_{\lambda t}(a)$$

This is a well-known problem that has a closed-form solution as follows:

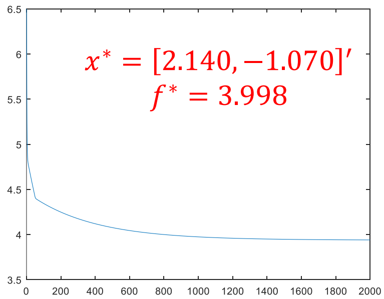
$$[S_{\lambda t}(x)]_i = \begin{cases} x_i - t\lambda & x_i > t\lambda \\ 0 & -t\lambda \le x_i \le t\lambda \\ x_i + t\lambda & x < -t\lambda \end{cases}$$

$$prox_t(x - t\nabla g(x)) = S_{\lambda t}((x + tA^T(y - Ax))$$

Proximal Gradient Descent - Example

- Let $f(x) = \frac{1}{2}||y - Ax||_2^2 + \lambda||x||_1$
- Take $y = \begin{bmatrix} -2 \\ 3 \end{bmatrix}$ $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ and $\lambda =$ 0.5. Also use constant t= 0.01

\# of iteration for achieving ϵ = 0.0001 : 991




```
A = [1 2;3 4]; y = [-2;3];
lambda = 0.5;t = 0.01;iterations = 2000;
x = zeros(2, iterations); L = zeros(1, iterations);
L(1) = 0.5*(y - A*x(:,1))' * (y - A*x(:,1)) +
lambda * norm(x(:,1), 1);
x_new = zeros(size(A, 2), 1); %initiate
for i = 2 : iterations
    x_current = x(:, i-1);
    grad_g = -A'*(y - A*x_current);
    x_new_gd = x_current - t * grad_g;
    for k = 1 : length(x_new_gd)
        if x_new_gd(k) > lambda * t; x_new(k) =
x_new_gd(k) - lambda * t;
        elseif x_new_gd(k) < -lambda * t; x_new(k)
= x_new_gd(k) + lambda * t;
        else; x_new(k) = 0; end
    end
    x(:, i) = x_new;
    L(i) = 0.5*(norm(y - A*x_new))^2  + lambda *
norm(x_new, 1);
end
```

Proximal Gradient Descent - Special Case

- Let $f(x)$ be a differentiable function. Use proximal gradient descent to find the minimizer of f.
- Because $f(x)$ is differentiable, then $h(z) = 0$. Therefore
$$prox_t(x) = x$$
- Then, the iterative procedure of proximal gradient descent is as follows,
  - Choose x_0 as an initial value
  - Iterate using the following formula
$$x_{k+1} = x_k - t_k \nabla f(x_k)$$
- The proximal gradient descent is the same as gradient descent for differentiable functions.
- Therefore, proximal gradient descent has the $O(\frac{1}{\varepsilon})$ convergence rate (same as GD).

Accelerated Proximal Gradient Descent

- Similar to gradient descent, the proximal GD can be accelerated.
- Let $f(x) = g(x) + h(x)$ where g is convex and differentiable, and h is convex but not differentiable.
- The accelerated proximal gradient descent is as follows:
  - Select an initial values x_0
  - Define $v = x_{k-1} + \frac{k-2}{k+1}(x_{k-1} - x_{k-2})$
  - Set $x_k = prox_{t_k}(v - t_k \nabla g(v))$
- v carries momentum from previous iterations.
- The convergence rate is same as the accelerated GD (i.e., $O(\frac{1}{\sqrt{\varepsilon}})$).

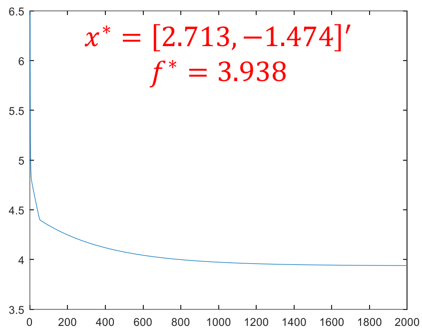
Algorithm 2 Accelerated Proximal Gradient Descent

```
Define prox_t(x)
Initialize x_{-1} and x_0
for k = 1 : K do
    Compute v = x_{k-1} + (k-2)/(k+1) (x_{k-1} - x_{k-2})
    Compute x_k = prox_{t_k}(v - t_k ∇g(v))
end for
```

Accelerated Proximal GD - Example

- Let $f(x) = \frac{1}{2}||y - Ax||_2^2 + \lambda||x||_1$
- Take $y = \begin{bmatrix} -2 \\ 3 \end{bmatrix}$ $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ and $\lambda =$ 0.5. Also use constant t= 0.01

# of iteration for achieving ϵ = 0.0001 : 101




```
A = [1 2;3 4]; y = [-2;3];
lambda = 0.5;t = 0.01;iterations = 2000;
x = zeros(2, iterations); L = zeros(1, iterations);
L(1) = 0.5*(y - A*x(:,1))' * (y - A*x(:,1)) + lambda *
norm(x(:,1), 1);
x_new = zeros(size(A, 2), 1); %initiate
for i = 3 : iterations
    x_current = x(:, i-1);
    x_prev = x(:, i - 2);
    v = x_current + ((i -  2)/ (i + 1)) * (x_current - x_prev);
    grad_g = -A'*(y - A*v);
    x_new_gd = v - t * grad_g;
    for k = 1 : length(x_new_gd)
        if x_new_gd(k) > lambda * t; x_new(k) = x_new_gd(k) -
lambda * t;
        elseif x_new_gd(k) < -lambda * t; x_new(k) =
x_new_gd(k) + lambda * t;
        else x_new(k) = 0; end
    end
    x(:, i) = x_new;
    L2(i) = 0.5*(norm(y - A*x_new))^2  + lambda * norm(x_new,
1);
end
```

Topics on High-Dimensional Data Analytics

Optimization Methods II

Kamran Paynabar, Ph.D.
Associate Professor
School of Industrial and Systems Engineering

Augmented Lagrangian Method and ADMM

Learning Objectives

- Describe Augmented Lagrangian Methods
- Explain Alternating Direction Method of Multipliers (ADMM)

Augmented Lagrangian Method

- Proximal gradient descent is useful for minimizing decomposable convex functions with a differentiable and a non-differentiable part.
- However, if $prox_t(x)$ is not easy to obtain, proximal gradient method may not be useful.
- For example, if a function is in the form of $f(x) = g(x) + h(x)$, but $h(x) = r(Ax)$, it may be difficult to use the proximal gradient descent method.
- Augmented Lagrangian method and Alternating direction method of multipliers (ADMM) are two methods that can be useful in these cases.

Augmented Lagrangian Method

- Assume we want to solve the following problem:

$$\min_x f(x) \qquad subject\ to\ Ax = b,$$

where $x \in R^d$ and A is a $m \times d$ matrix and $b \in R^m$ and $f(x)$ is a convex function

- The main idea is to solve a constraint problem by solving a series of unconstraint problems.
- The augmented Lagrangian objective function is given by ($\rho > 0$ is a parameter)

$$L(x, u; \rho) = f(x) + \underbrace{u^T(Ax - b)}_{\text{Lagrangian}} + \underbrace{\frac{\rho}{2}||Ax - b||_2^2}_{\text{Augmented}}$$

- The augmentation gives the strong convexity properties without changing the minimizer of the objective function.

Augmented Lagrangian Method

$$L(x, u; \rho) = f(x) + u^T(Ax - b) + \frac{\rho}{2}||Ax - b||_2^2$$

The update of the Augmented Lagrangian method (a.k.a. method of multipliers) is as following:

$$x^+ = \arg\min_x f(x) + u^T Ax + \frac{\rho}{2}||Ax - b||_2^2$$

$$u^+ = u + \rho(Ax^+ - b)$$

- One can use acceleration and backtracking (on u) as before for this algorithm.

Algorithm 3 Augmented Lagrangian Method

```
Select ρ and ϵ
Initialize u_0
for k = 1 : K do
    Compute x_k = argmin_x { f(x) + u_{k-1}^T Ax + (ρ/2) ||Ax - b||_2^2 }
    Compute u_k = u_{k-1} + ρ(Ax_k - b)
    if ||Ax - b||_2 < ϵ then
        Break
    end if
end for
```

Augmented Lagrangian Method - Example

Minimize $f(x) = \frac{1}{2}(x_1^2 + x_2^2)$ where $x = (x_1, x_2)^T$

Subject to $x_1 = 1$

The steps for the Augmented Lagrangian methods are as following:

- Initiate x_0 and u_0, then or k=1,2,…

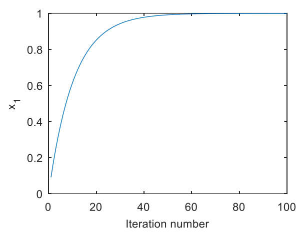
  (1) $x_k = \arg\min_x \frac{1}{2}(x_1^2 + x_2^2) + u_{k-1}(x_1 - 1) + \frac{\rho}{2}(x_1 - 1)^2$

  (2) $u_k = u_{k-1} + \rho(x_1 - 1)$

- By taking derivative from (1) we can derive,

  $x_1 = \frac{\rho - u_{k-1}}{1+\rho}$ and $x_2 = 0$

```
iterations = 100;
u = 0;
rho = 0.1;
x_2 = 0; % x_2 remains fixed
as we derived.
for i = 1 : iterations
   x_1 = (rho - u)/(1+ rho);
   x_1_record(i) = x_1;
   u = u + rho* (x_1 - 1);
   L(i) = 0.5* x_1^2 + 0.5 *
x_2^2;
end
```



Alternating Direction Method of Multipliers (ADMM)

Assume we want to solve the following problem:

$$\min_{x,z} f(x) + g(z) \qquad subject\ to\ Ax + Bz = c$$

Similar to the Augmented Lagrangian method, we augment the objective function as the following:

$$\min_{x,z} f(x) + g(z) + \frac{\rho}{2}||Ax + Bz - c||_2^2 \qquad subject\ to\ Ax + Bz = c$$

- The augmented term does not change the objective, as its value is always zero due to the constraints.
- Define the augmented Lagrangian function (for a given ρ) as following:

$$L(x, z, u; \rho) = \underbrace{f(x) + g(z) + u^T(Ax + Bz - c)}_{\text{Lagrangian}} + \underbrace{\frac{\rho}{2}||Ax + Bz - c||_2^2}_{\text{Augmented}}$$

Alternating Direction Method of Multipliers (ADMM)

Define the augmented Lagrangian function (for a given ρ) as following:

$$L(x,z,u;\rho) = f(x) + g(z) + u^T(Ax + Bz - c) + \frac{\rho}{2}\left|\left|Ax + Bz - c\right|\right|_2^2$$

Then the ADMM updates are as following:

- Initiate z_0 and u_0 and for $k = 1,2,\ldots$ iterate using the following:

$$x_k = \arg\min_x L(x, z_{k-1}, u_{k-1} \; ;\rho)$$

$$z_k = \arg\min_z L(x_k, z, u_{k-1} \; ;\rho)$$

$$u_k = u_{k-1} + \rho(Ax_k + Bz_k - c)$$

- Note that original augmented Lagrangian combines the updates of x, z:

$$(x_k, z_k) = \arg\min_{x,z} L(x, z, u_{k-1} \; ;\rho)$$

Scaled Form of ADMM

If we define $w = \frac{u}{\rho}$ then the scaled form of the ADMM is as following:

$$L(x,z,w;\rho) = f(x) + g(z) + \frac{\rho}{2}\left|\left|Ax + Bz - c + w\right|\right|_2^2 + \frac{\rho}{2}\left|\left|w\right|\right|_2^2$$

With the following updates:

$$x_k = \arg\min_x f(x) + \frac{\rho}{2}\left|\left|Ax + Bz_{k-1} - c + w_{k-1}\right|\right|_2^2$$

$$z_k = \arg\min_z g(z) + \frac{\rho}{2}\left|\left|Ax_k + Bz - c + w_{k-1}\right|\right|_2^2$$

$$w_k = w_{k-1} + Ax_k + Bz_k - c$$

Algorithm 4 ADMM

```
Initialize u_0 and z_0
for k = 1 : K do
    Compute x_k = arg min_x { f(x) + ρ/2 ||Ax + Bz_{k-1} - c + w_{k-1}||_2^2 }
    Compute z_k = arg min_z { g(z) + ρ/2 ||Ax_k + Bz - c + w_{k-1}||_2^2 }
    Compute w_k = w_{k-1} + Ax_k + Bz_k - c
end for
```

ADMM- Example

Minimize $h(x) = \frac{1}{2}(x_1^2 + x_2^2)$

Subject to $x_1 + x_2 = 1$

Here, $f(x) = 0.5x_1^2$ and $g(z) = 0.5x_2^2$ and $A = 1; B = 1; c = 1$.

ADMM updates:

$$x_{1,k} = \arg\min_{x_1} 0.5x_1^2 + \frac{\rho}{2}\left(x_1 + x_{2,k-1} - 1 + w_{k-1}\right)^2$$

$$x_{2,k} = \arg\min_{x_2} 0.5x_2^2 + \frac{\rho}{2}\left(x_{1,k} + x_2 - 1 + w_{k-1}\right)^2$$

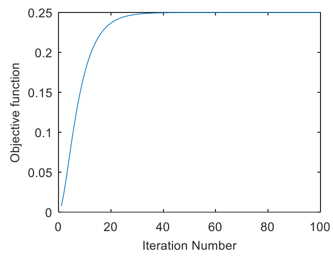
$$w_k = w_{k-1} + x_{1,k} + x_{2,k} - 1$$

Form the first two equations:

$$x_{1,k} = \frac{-\rho(x_{2,k-1} - 1 + w_{k-1})}{1 + \rho} \text{ and } x_{2,k} = \frac{-\rho(x_{1,k-1} - 1 + w_{k-1})}{1 + \rho}$$

$$x_k = \arg\min_x f(x) + \frac{\rho}{2}\left|\left|Ax + Bz_{k-1} - c + w_{k-1}\right|\right|_2^2$$

```
iterations = 100;
rho = 0.1;
w = 0; % intial value
x_2 = 0; % intial value.
for i = 1 : iterations
   x_1 = -rho*(x_2 - 1 + w)/(1+ rho);
   x_2 = -rho*(x_1 - 1 + w)/(1+ rho);
   w = w + x_1 + x_2 -1;
   L(i) = 0.5* x_1^2 + 0.5 * x_2^2;
end
plot(L); ylabel('Objective function');
xlabel('Iteration number')
```



ADMM and Proximal Gradient Descent

- Recall that the objective function of the proximal gradient descent was in the following form:

$$Minimize\ f(x), \text{where } f(x) = g(x) + h(x).$$

We can write the above objective as

$$Minimize\ g(x) + h(z)\ \ subject\ to\ x = z$$

Which gives an objective function in the form suitable for ADMM.

- What if $h(x) = r(Ax)$? In this situation proximal gradient method is difficult to use but ADMM can be used as follows:

$$Minimize\ g(x) + r(z)\ \ subject\ to\ Ax = z$$

Some Notes on ADMM

In practice

- ADMM can reach relatively accurate results in a few iterations. However, for highly-accurate results, ADMM requires many iterations.
- Selection of ρ influences the convergence of the algorithm:
  - If ρ is too large, then we are not appropriately minimizing the objective function
  - If ρ is too small, we may end up with infeasible solutions.

Topics on High-Dimensional Data Analytics

Optimization Methods II

Kamran Paynabar, Ph.D.
Associate Professor
School of Industrial and Systems Engineering

Coordinate Descent Algorithm

Learning Objectives

- Define decomposable functions
- Use coordinate descent algorithm
- Apply block coordinate descent algorithm

Decomposable Functions - Revisit

- We introduced decomposable functions that can be written as differentiable and non-differentiable parts.
- These functions, however, may or may not be decomposed in their coordinates.
- Suppose that we want to solve the following problem:

$$Minimize f(x) = g(x) + h(x),$$

where $h(x) = \sum_i h_i(x_i)$, $g(x)$ is convex and differentiable and each h_i is convex.

- As can be seen, $h(x) = \sum_i h_i(x_i)$ is decomposable with respect to the coordinates.

Decomposable Functions - Revisit

- Suppose that we want to solve the following problem:

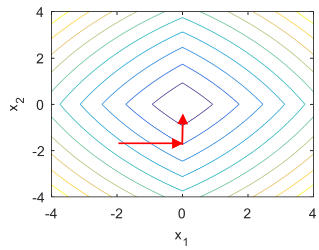
$$Minimize f(x) = g(x) + h(x),$$

where $h(x) = \sum_i h_i(x_i)$, $g(x)$ is convex and differentiable and each h_i is convex.

Example:

$$f(x) = \frac{1}{2}||y - Ax||_2^2 + ||x||_1 = \frac{1}{2}||y - Ax||_2^2 + |x_1| + |x_2|$$

Let $y = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $A = \begin{bmatrix} 0.3 & 0 \\ 0 & 0.3 \end{bmatrix}$



- When minimizing along each direction, the objective function is minimized

This results in an algorithm called coordinate descent

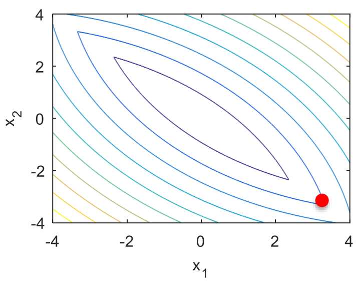
Decomposable Functions - Revisit

- What if the non-differentiable part ($h(x)$) is not decomposable into its coordinates?

Example:

Let $f(x) = \frac{1}{2}||y - Ax||_2^2 + |x_1 + x_2|$

Take $y = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $A = \begin{bmatrix} 0.3 & 0 \\ 0 & 0.3 \end{bmatrix}$



At the red point, we cannot minimize along one direction only.

- Therefore, separability of the nonsmooth part into its coordinates (or a block of coordinates) is a requirement to be able to solve the problem by minimizing along each direction.

Coordinate Descent Algorithm

Suppose that we want to solve the following problem:

$$Minimize\ f(x) \quad and\ x \in R^n$$

Where $f(x) = g(x) + h(x)$ and $h(x) = \sum_i h_i(x_i)$. Here, $g(x)$ is convex and differentiable and each h_i is convex.

Then we can find the minimum by minimizing along each coordinate as following:

- Initiate $x = x_0$ and the for each coordinate i and for iternations $k = 1,2, \ldots$ solve

$$x_i^k = \arg\min_{x_i} f(x_1^k, x_2^k, \ldots, x_i, x_{i+1}^{k-1}, \ldots, x_n^{k-1})$$

- We always use most updated results from other coordinates.

Coordinate Descent Algorithm

Initiate $x = x_0$ and the for each coordinate i and for iternations $k = 1,2, \ldots$ solve

$$x_i^k = \arg\min_{x_i} f(x_1^k, x_2^k, \ldots, x_i, x_{i+1}^{k-1}, \ldots, x_n^{k-1})$$

Algorithm 5 Coordinate Descent

```
1: Initialize x_0 ∈ R^n
2: for k = 1 : K do
3:     for i = 1 : n do
4:         Compute x_i^k = arg min_{x_i} f(x_1^k, x_2^k, ··· x_{i-1}^k, x_i, x_{i+1}^{k-1}, ··· , x_n^{k-1})
5:     end for
6: end for
```

- If the nonsmooth part is decomposable into groups of coordinates, we should minimize the function along each block of coordinates. The resulting algorithm is known as block coordinate descent.

Coordinate Descent Algorithm - Example

Minimize $f(x) = \frac{1}{2}\left||y - Ax|\right|_2^2 + 0.5|x_1| + 0.5|x_2|$

Let $y = \begin{bmatrix} -2 \\ 3 \end{bmatrix}$ $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

Using coordinate descent algorithm, in each iteration we have,

$$x_1^k = \arg\min_{x_1} \frac{1}{2}\left|\left|y - A(x_1, x_2^{k-1})^T\right|\right|_2^2 + 0.5|x_1| + 0.5|x_2^{k-1}|$$

$$x_2^k = \arg\min_{x_2} \frac{1}{2}\left|\left|y - A(x_1^k, x_2)^T\right|\right|_2^2 + 0.5\left|x_1^k\right| + 0.5|x_2|$$

Coordinate Descent Algorithm - Example

$$x_1^k = \arg\min_{x_1} \frac{1}{2}\left|\left|y - A(x_1, x_2^{k-1})^T\right|\right|_2^2 + 0.5|x_1| + 0.5|x_2^{k-1}|$$

$$x_2^k = \arg\min_{x_2} \frac{1}{2}\left|\left|y - A(x_1^k, x_2)^T\right|\right|_2^2 + 0.5\left|x_1^k\right| + 0.5|x_2|$$

The solution to each of these is soft-thresholding:

Let A_i be the i^{th} column of the matrix A, and define $\gamma_i = \frac{0.5}{||A_i||_2^2}$ then,

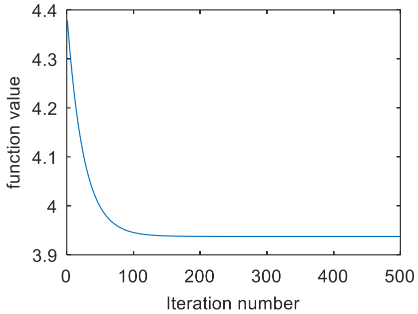
$$x_1^k = S_{\gamma_1}\left(\frac{A_1^T(y - A_2 x_2^{k-1})}{A_1^T A_1}\right) \text{ and } x_2^k = S_{\gamma_2}\left(\frac{A_2^T(y - A_1 x_1^k)}{A_2^T A_2}\right)$$

$$\text{where } S_{\gamma_i}(a) = \begin{cases} a - \gamma_i & a > \gamma_i \\ 0 & -\gamma_i \le a \le \gamma_i \\ a + \gamma_i & a < -\gamma_i \end{cases}.$$

Coordinate Descent Algorithm – Example Continued

```
A = [1 2;3 4]; % this is same as the gradient of g
y = [-2;3];
lambda = 0.5; x(1) = 0; x(2) = 0;e(1)=1;L(1)=5;k=1;
while e > 0.0001
      k=k+1;
    for i = 1 : 2
        Ai = A(:,i);
        A_i = A(:, 2-i+1);
        gamma = lambda/(Ai'*Ai);
        z = Ai'*(y - A_i * x(2-i+1))./(Ai'*Ai);
        x(i) = softhres(z, gamma);
    end
    L(k) = 0.5*(norm(y - A*x'))^2  + lambda * norm(x, 1);
      e(k) = L(k-1) - L(k);
end

function s = softhres(x, gamma)
    if x > gamma
        s = x - gamma;
    elseif x < -gamma
        s = x + gamma;
    else
        s = 0;
    end
end
```



# of iterations for achieving ϵ = 0.0001 : 131

$x^* = [2.595, -1.392]'$

$f^* = 3.940$

APGD: # of iterations is 101

$x^* = [2.713, -1.474]'$

$f^* = 3.938$

PGD: # of iterations is 991

$x^* = [2.140, -1.070]'$

$f^* = 3.998$