The Alternating Direction Method of Multipliers

With Adaptive Step Size Selection

Peter Sutor, Jr.

Project Advisor: Professor Tom Goldstein

December 2, 2015

The Dual Problem

- Consider the following problem (*primal problem*):
  $$\min_x(f(x)) \text{ subject to } Ax = b.$$
- Important components of this problem:
  1. The Lagrangian: $L(x, y) = f(x) + y^T(Ax - b)$
     - We refer to the original x variable as the *primal variable* and the y variable as the *dual variable*.
  2. Dual function: $g(y) = \inf_x(L(x, y))$
     - New function made purely out of the dual variable.
     - Gives a lower bound on the objective value.
  3. Dual problem: $\max_{y \geq 0}(g(y))$
     - The problem of finding the best lower bound.
- End goal: recover $x^* = \arg\min_x(L(x, y^*))$, where x^* and y^* are corresponding optimizers.

Methods Discussed

- Dual Ascent Method (DAM): General gradient type method. Uses Lagrangian and dual variable updates to solve optimization problem.
- Method of Multipliers (MM): Add to Lagrangian penalty term: $\rho/2||Ax - b||_2^2$.
  1. Very robust method.
  2. Penalty prevents decomposing the problem.
- Dual Decomposition (DD): Decompose dual variable, update primal components in parallel, then update dual.
  1. Need separable function.
  2. Can be slow to converge.

The Alternating Direction Method of Multipliers (ADMM)

- Finds a way to combine advantages of DD and MM.
  - Robustness of the Method of Multipliers.
  - Supports Dual Decomposition $\rightarrow$ parallel x-updates possible.
- Problem form: (where f and g are both convex)

$$\min\left(f(x) + g(z)\right) \text{ subject to } Ax + Bz = c,$$

- Objective is separable into two sets of variables.
- ADMM defines a special Augmented Lagrangian to enable decomposition: $(r = Ax + Bz - c,\ u = y/\rho)$

$$\begin{aligned} L_\rho(x,z,y) &= f(x) + g(z) + y^T(r) + \frac{\rho}{2}||r||_2^2 \\ &= f(x) + g(z) + (\rho/2)||r + u||_2^2 - const \\ &= L_\rho(x,z,u) \end{aligned}$$

ADMM Algorithm

- Repeat for $k = 0$ to specified n, or until convergence:
  1. $x^{(k+1)} := \arg\min_x (L_\rho(x, z^{(k)}, u^{(k)}))$
  2. $z^{(k+1)} := \arg\min_z (L_\rho(x^{(k+1)}, z, u^{(k)}))$
  3. $u^{(k+1)} := u^{(k)} + (Ax^{(k+1)} + Bz^{(k+1)} - c)$
- Recall the *proximal operator*: (with $v = Bz^{(k)} - c + u^{(k)}$)

$$\mathbf{prox}_{f,\rho}(v) := \arg\min_x (f(x) + (\rho/2)||Ax + v||_2^2)$$

- If $g(z) = \lambda||z||_1$, then $\mathbf{prox}_{g,\rho}(v)$ is computed by soft-thresholding: (with $v = Ax^{(k+1)} - c + u^{(k)}$)

$$z_i^{(k+1)} := sign(v_i)(|v_i| - \lambda)_+$$

In this project...

- Our goal is to make ADMM easier to use in practice: upload A, B, and c, then run appropriate function, or supply proximal functions for f and g and run general ADMM.
- Maximizing ADMM's potential means tweaking parameters such as step size ρ and more.
- Hope to create a comprehensive library for general ADMM use.
  - Generalized ADMM functionality (with customizable options).
  - Adaptive step-size selection.
  - Ready to go optimized functions for problems ADMM is most used for (with customizable options).
  - High performance computing capabilities (MPI).
  - Implementations in Python and Matlab.

Goals for Fall Semester

1. Implement/test/validate a general ADMM function with fully customizable options for users.
   - Convergence checking of proximal operators.
   - Stopping conditions.
   - Complete run-time information.
2. Implement/test/validate the following 3 ADMM solvers:
   - LASSO Problem: Least absolute shrinkage and selection operator, a regularized form of the Least Squares Problem.
   - Total Variation Minimization: Minimize overall variation in a given signal.
   - Linear Support Vector Machines (SMVs): Classifiers where classes are linearly separable.
3. Devise an efficient adaptive step-size selection algorithm for ADMM.

The Progress So Far

- General ADMM and the three solvers are as finished as they can be at this point.
- Testing and validation code has also been finished.
- User options, stopping conditions and convergence checking are also finished. More can be done here.
- Adaptive ADMM has been programmed and studied. Some issues here (discussed later).

Stopping Conditions

- Primal (p) and Dual (d) residuals in ADMM at step $k+1$:
  - $p^{k+1} = Ax^{k+1} + Bz^{k+1} - c$
  - $d^{k+1} = \rho A^T B(z^{k+1} - z^k)$
- Reasonable stopping criteria: $||p^k||_2 \leq \epsilon^{pri}$ and $||d^k||_2 \leq \epsilon^{dual}$.
- Many ways to choose these tolerances.
- One common example, where $p \in \mathbb{R}^{n_1}$ and $d \in \mathbb{R}^{n_2}$:
  - $\epsilon^{pri} = \sqrt{n_1}\epsilon^{abs} + \epsilon^{rel} \max(||Ax^k||_2, ||Bz^k||_2, ||c||_2)$
  - $\epsilon^{dual} = \sqrt{n_2}\epsilon^{abs} + \epsilon^{rel}||A^T y^k||_2$

  where ϵ^{abs} and ϵ^{rel} are chosen constants referred to as *absolute* and *relative* tolerance.

Convergence Checking

- Paper by He and Yuan gives way of constructing monotonically decreasing residual norms:

$$||w^k - w^{k+1}||_H^2 \leq ||w^{k-1} - w^k||_H^2$$

where $w^i = \begin{bmatrix} x^i \\ z^i \\ \rho u^i \end{bmatrix}$ and $H = \begin{bmatrix} G & 0 & 0 \\ 0 & \rho B^T B & 0 \\ 0 & 0 & I_m/\rho \end{bmatrix}$

- The H-norm squared can be easily calculated. We then expect: (e.g., $\epsilon = 10^{-16}$, for $k \geq 3$)

$$||w^{k-1} - w^k||_H^2 - ||w^{k-1} - w^{k-1}||_H^2 \leq \epsilon$$

- User can specify tolerance ϵ; algorithm stops if tolerance is broken as convergence is compromised.

A Model Problem

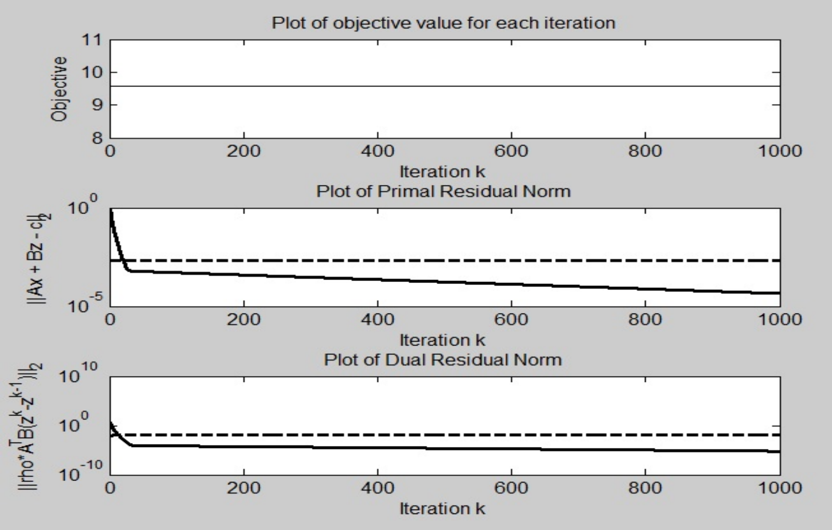
- Consider: $\arg\min_x(||Ax-b||_2^2 + ||Cx-d||_2^2)$, $A, C \in \mathbb{R}^{n\times n}$.
- By setting derivative to 0 and solving, exact solution is $x = (A^TA + C^TC)^{-1}(A^Tb + C^Td)$.
- In ADMM form: (with $f(x) = ||Ax-b||_2^2$, $g(z) = ||Cz-d||_2^2$)

$$\arg\min_x(||Ax-b||_2^2 + ||Cz-d||_2^2), \text{ subject to } x - z = 0$$

- $L_\rho(x,z,u) = f(x) + g(z) + \rho/2||x-z+u||_2^2$
- Proximal operators:
  1. $\mathbf{prox}_{f,\rho}(x, z^k, u^k) = (2A^TA + \rho I_n)^{-1}(2A^Tb + \rho(z^k - u^k))$
  2. $\mathbf{prox}_{g,\rho}(x^{k+1}, z, u^k) = (2C^TC + \rho I_n)^{-1}(2C^Td + \rho(x^{k+1} + u^k))$

Model Problem: Example Output

```
>> admm_test
For n = 2^1, test 1 -- Relative error acceptable: 0
For n = 2^2, test 1 -- Relative error acceptable: 5.234732e-16
For n = 2^3, test 1 -- Relative error acceptable: 1.179740e-16
For n = 2^4, test 1 -- Relative error acceptable: 5.318879e-16
For n = 2^5, test 1 -- Relative error acceptable: 1.305104e-16
For n = 2^6, test 1 -- Relative error acceptable: 2.175907e-12
For n = 2^7, test 1 -- Relative error acceptable: 6.699444e-07
For n = 2^8, test 1 -- RELATIVE ERROR UNACCEPTABLE: 1.235201e-03; 2.269872e+01 vs. true 2.267071e+01
For n = 2^9, test 1 -- RELATIVE ERROR UNACCEPTABLE: 8.463742e-03; 4.152521e+01 vs. true 4.117671e+01
Average time for size 2^1: 0.092002 seconds.
Average time for size 2^2: 0.089818 seconds.
Average time for size 2^3: 0.12141 seconds.
Average time for size 2^4: 0.11181 seconds.
Average time for size 2^5: 0.16372 seconds.
Average time for size 2^6: 0.25715 seconds.
Average time for size 2^7: 0.62841 seconds.
Average time for size 2^8: 1.8398 seconds.
Average time for size 2^9: 9.6431 seconds.
2 UNACCEPTABLE ERROR(S) FOR TOLERANCE 0.001, for 1000 iterations!
>>
```


Plot of objective value for each iteration
Objective
11
10
9
8
0
200
400
600
800
1000
Iteration k
Plot of Primal Residual Norm
||Ax + Bz - c||2
10^0
10^-5
Iteration k
Plot of Dual Residual Norm
||rho*A^TB(z^k-z^k-1)||2
10^10
10^0
10^-10
Iteration k

Breaking the Convergence Check

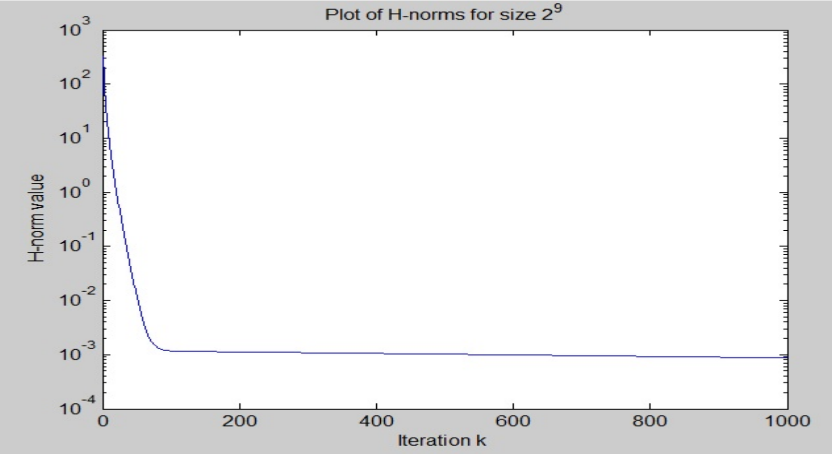
- Suppose we change the x-update in the model problem:
  - Old: $\mathbf{prox}_{f,\rho}(x, z^k, u^k) = (2A^TA + \rho I_n)^{-1}(2A^Tb + \rho(z^k - u^k))$
  - New: $\mathbf{prox}_{f,\rho}(x, z^k, u^k) = (2A^TA + \rho)^{-1}(2A^Tb + \rho(z^k - u^k))$
- Then, ADMM should not converge, as this is not convex.
- The H-norms for the original proximal operator are monotonically decreasing, however.

```
>> admm_test
Error using admm (line 268)
Iteration 3: H norms not converging to given relative tolerance: 3.253359e+06 vs. tol. 1.000000e-15

Error in admm_test (line 62)
        [results] = admm(proxf, proxg, options);


>>
```

H-norms on Model Problem



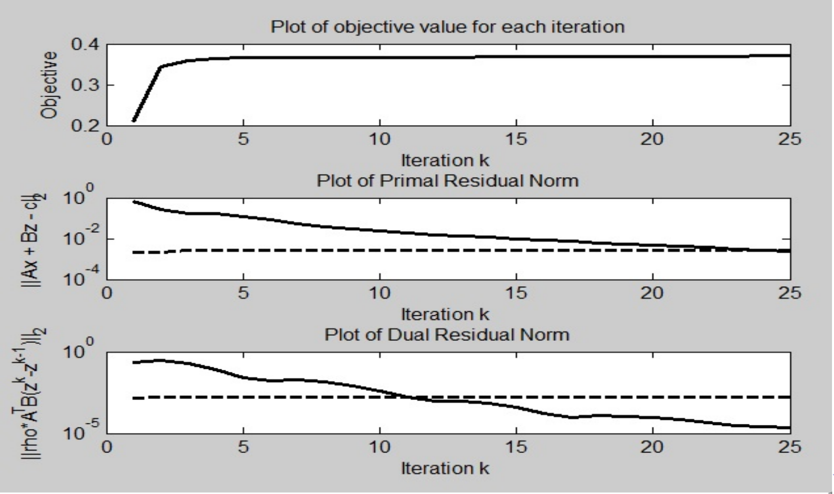
LASSO Problem

- Standard LASSO formulation:

$$\min_x (1/2||Dx - b||_2^2 + \lambda||x||_1)$$

- Can use transpose reduction. We note that $1/2||Dx - b||_2^2 = 1/2x^T(D^TD)x - x^TD^Tb + 1/2||b||_2^2$
- Now, a central server needs only D^TD and D^Tb. For tall, large D, D^TD has much fewer entries.
- Note that: $D^TD = \sum_i D_i^TD_i$ and $D^Tb = \sum_i D_i^Tb_i$.
- Now each server need only compute local components and aggregate on a central server.
- Once D^TD and D^Tb are computed, solve with ADMM.

Sample LASSO Ouput



Unwrapped ADMM (Goldstein)

- Consider the problem $\min(g(Dx))$, where g is convex and $D \in \mathbb{R}^{m \times n}$ is a large, distributed data matrix.
- In "unwrapped" ADMM form: $min(g(z))$ subject to $Dx - z = 0$ $(f(x) = 0)$. The z update is typical, but special x updated for distributed data: $D^+(z^k - u^k)$, where $D^+ = (D^T D)^{-1} D^T$.
- If g is decomposable, each component in z update is decoupled. Analytical solution or look-up table is possible.
- As $D = [D_1^T, ..., D_n^T]^T$, x update can be rewritten as:

$$x^{k+1} = D^+(z^k - u^k) = W \sum_i D_i(z_i^k - u_i^k)$$

- Note that $W = (\sum_i D_i^T D_i)^{-1}$. Each vector $D_i(z_i^k - u_i^k)$ can be computed locally, while only multiplication by W occurs on central server.

Linear SVMs

- General Form: $\min(1/2||x||^2 + Ch(Dx))$, C a regularization parameter. The function h is the "hinge loss" function: $h(z) = \sum_{k=1}^{M} \max(1 - \ell_k z_k, 0)$.
- Unwrapped ADMM can solve this problem, along with the "zero-one loss" function.
- For hinge loss: $z^{k+1} = Dx + u + \ell \max(\min(1 - v, C/\rho), 0)$
- For 0-1 loss: $z^{k+1} = \ell \mathbb{I}(v \geq 1 \text{ or } v < (1 - \sqrt{2C/\rho}))$
- Here, $v = \ell(Dx + u)$

Results for Hinge vs. 0-1 Loss on MNIST dataset

250 Iterations

Error Percentages: 600 training, 100 test samples.

Digit	Hinge (Train)	0-1 (Train)	Hinge (Test)	0-1 (Test)
0	2.0000	2.0000	13.0000	13.0000
1	0.8333	0.8333	9.0000	9.0000
2	3.1667	3.3333	22.0000	20.0000
3	2.5000	2.1667	21.0000	22.0000
4	3.3333	3.1667	12.0000	12.0000
5	4.6667	4.6667	18.0000	18.0000
6	1.8333	1.8333	12.0000	12.0000
7	2.5000	2.5000	23.0000	23.0000
8	4.8333	4.6667	18.0000	20.0000
9	3.5000	3.6667	30.0000	28.0000

Elapsed time is 10.735045 seconds.

250 Iterations

Error Percentages: 6000 training, 1000 test samples.

Digit	Hinge (Train)	0-1 (Train)	Hinge (Test)	0-1 (Test)
0	2.4667	2.8167	4.3000	4.7000
1	2.0833	2.6000	4.3000	4.7000
2	3.9333	4.0333	8.3000	7.7000
3	5.1833	4.5500	9.0000	8.0000
4	3.6333	4.2667	7.8000	8.8000
5	5.5833	4.5833	9.9000	8.8000
6	2.4833	3.2833	5.2000	6.1000
7	3.0500	3.7000	5.6000	6.4000
8	13.2500	8.5833	16.5000	13.0000
9	8.2667	7.9333	12.7000	12.5000

Elapsed time is 102.358839 seconds.

250 Iterations

Error Percentages: 60000 training, 10000 test samples.

Digit	Hinge (Train)	0-1 (Train)	Hinge (Test)	0-1 (Test)
0	2.9850	3.1417	3.0100	3.2400
1	3.0050	3.5200	2.7400	3.2200
2	5.9383	5.1150	5.7300	5.2900
3	7.4017	6.3633	7.4500	6.6100
4	5.2450	5.3500	5.9700	6.2100
5	7.4867	6.0067	7.5000	6.0600
6	3.7483	3.8583	4.1300	4.2600
7	4.2467	4.4667	4.2800	4.6800
8	15.0200	10.5783	15.3800	11.3700
9	10.9150	10.0067	11.0600	10.1800

Elapsed time is 1016.946927 seconds.

2000 Iterations

Error Percentages: 12000 training, 2000 test samples.

Digit	Hinge (Train)	0-1 (Train)	Hinge (Test)	0-1 (Test)
0	2.1000	1.8750	3.6500	3.0000
1	1.5667	1.7583	2.8000	2.9000
2	5.1167	2.7583	5.9500	4.2000
3	7.1000	3.6833	7.1000	4.7500
4	4.1750	3.2333	6.1000	5.0500
5	5.8583	3.3583	6.9000	4.5000
6	2.4667	1.9000	4.0000	3.6000
7	3.2917	2.8833	4.2500	4.4500
8	13.5750	6.8000	16.2500	10.1000
9	9.0083	6.5750	10.1500	7.6000

Elapsed time is 1694.761875 seconds.

Adaptive Step-sizes

- Strategy for adaptive step-sizes:
  - According to Esser's paper, the Douglas Rachford Splitting Method (DRSP) and ADMM are equivalent. ADMM is DRSP applied to the dual problem

$$max_{u \in \mathbb{R}^d} (inf_{x \in \mathbb{R}^{m_1}, z \in \mathbb{R}^{m_1}} (L(x, z, u)))$$

  - So ADMM is equivalent to finding u such that $0 \in \psi(u) + \phi(u)$, where $\psi(u) = B\partial g^*(B^T u) - c$ and $\phi(u) = A\partial f^*(A^T u)$.
  - Form residuals equal to $\psi(u^k) + \phi(u^k)$. Interpolate with last residual over stepsize ρ.
  - Solve this as least squares problem - closed form solution for optimal ρ!
- Hard to compute these residuals. If either f or g is strictly convex, can find closed form solution for either ϕ or ψ.

Issues with Adaptive Step-size Selection

- Various attempts gave step-sizes that often converged to high values. These negatively impact convergence.
- If the value of ρ does not explode, actually can get faster convergence!
- Still looking for a way to stabilize step-sizes.

```
Iteration 74: rho = 3.1008
Iteration 75: rho = 7.8428
Iteration 76: rho = 15.2461
Iteration 77: rho = 37.4974
Iteration 78: rho = 68.1072
Iteration 79: rho = 166.6155
Iteration 80: rho = 248.9479
Iteration 81: rho = 509.6552
Iteration 82: rho = 454.6806
Iteration 83: rho = 1369.1802
Iteration 84: rho = 145.9018
Iteration 85: rho = 52.3052
Iteration 86: rho = 30.4316
Iteration 87: rho = 68.4639
Iteration 88: rho = 75.2073
Iteration 89: rho = 646.4078
Iteration 90: rho = 349.0421
Iteration 91: rho = 1133.7667
Iteration 92: rho = 1902.1342
Iteration 93: rho = 28235.6798
Iteration 94: rho = 321928.8644
Iteration 95: rho = 46418182.0565
Iteration 96: rho = 62177105891.6655
Iteration 97: rho = 1.196223850308679e+16
```

Project Schedule

- **End of Fall Semester Goals:**
  - **End of October:** Implement generic ADMM, solvers for the Lasso problem, TV Minimization, and SVMs.
  - **Early November:** Implement scripts for general testing, convergence checking, and stopping condition strategies.
  - **Early December:** Finalize bells and whistles on ADMM options. Compile testing/validation data.
  - **End of November:** Implement a working adaptive step-size selection algorithm.
- **Spring Semester Goals:**
  - **End of February:** Implement the full library of standard problem solvers.
  - **End of March:** Finish implementing MPI in ADMM library.
  - **End of April:** Finishing porting code to Python version.
  - **Early May:** Compile new testing/validation data.

References

- S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers", *Foundations and Trends in Machine Learning*, vol. 3, no.1, pp. 1-122, 2010.
- T. Goldstein, G. Taylor, K. Barabin, and K. Sayre, "Unwrapping ADMM: Efficient Distributed Computing via Transpose Reduction", *CoRR*, vol. abs/1504.02147, 2015.
- E. Esser, *Applications of Lagrangian-Based Alternating Direction Methods and Connections to Split Bregman*, April 2009.
- B. He, X. Yuan, "On non-ergodic rate of Douglas-Rachford alternating direction method of multipliers," *Numerishe Mathematik*, vol. 130, iss. 3, pp. 567-577, 2014.
- H. Everett, "Generalized Lagrange multiplier method for solving problems of optimum allocation of resources," *Operations Research*, vol. 11, no. 3, pp. 399-417, 1963.

Thank you! Any questions?