به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

درس بهینه‌سازی توزیع‌شده چند عامله

گزارش پروژه نهایی

مژده کربلایی مطلب ۸۱۰۱۹۶۰۷۴

کیانا نوروزی ۸۱۰۱۹۶۳۳۷

بهمن ماه ۱۳۹۷

بخش۱: بیان مسئله

یک شبکه‌ی کاملا متصل شامل J نود را در نظر بگیرید که پیام $\tilde{x} \in R^n$ به هرکدام توسط یک نود مرکزی منتقل می‌گردد.

فرض می‌شود که هر J نود قادر به انتقال اطلاعات خود به یکدیگر می باشند.و تمام لینک‌های بین نودها ایده آل در نظر گرفته است و غیر قابل تغییر با زمان می باشند. هر نود J نسخه‌ی تغییر یافته‌ی پیام $v_j \in R^{m\times 1}$ را دریافت می‌نماید به طوری که $m \geq n$ می باشد و x_j به صورت زیر مدل می گردد.

$$v_j = U_j\tilde{x} + n_j$$

که به $U_j \in R^{m\times n}$ ماتریس اختلال اطلاق می‌شود و n_j نویز گوسی اضافه‌شونده با میانگین صفر و واریانس σ_n^2 است.

از این مسله میتوان در سیستم‌های شبکه‌ای متعددی اعم از: سیستم‌های چند انتنه، شبکه‌های مخابراتی چندکاربره و شبکه‌های حس‌گر استفاده کرد.

تخمین $\tilde{x}$ به صورت محلی برای هر نود منجر به رفتاری غیرمطلوب می‌شود، که پیام مخابره شده یکسان $\tilde{x}$، در هر نود به صورت متفاوتی تخمین زده میشود. بنابراین می‌بایست $\tilde{x}$ را درست تخمین زد.

وهمچنین اگر شرایط $m \geq n$ با داشتن ماتریس بزرگ U_j براورده نشود، یک نود j به دلیل سیستم معادلات خطی غیرقابل حل قادر به بازیابی $\tilde{x}$ نخواهد بود. اگر فرض کنیم که هر نود j قادر به ساختن یک اطلاعات سراسری از همه‌ی مشاهدات بردار v_j و U_j می‌باشد، می‌توانیم به یک کیفیت تخمین بهینه دست پیدا کنیم و مطمئن شویم که یک پیام یکسان در تمام نودها تخمین زده می‌شود. در این حالت از حداقل مربعات[1] برای تضمین یکسان بودن تخمین در هر نود استفاده می‌شود.

در این پروژه ما قصد داریم تا این مسله را به فرم توزیع شده و همچنین غیرمتمرکز بیان کنیم و نشان دهیم که ADMM قادر به دست‌یابی به پاسخ بهینه مسله خواهد بود که این جواب بسیار نزدیک به تخمین‌های متمرکز روش حداقل مربعات خواهد بود.

مسله‌ی تخمین $\tilde{x}$ به وسیله‌ی تمام نودها وبا تابع هدف حداقل کردن مجموع خطاهای تخمین به صورت زیر قابل بیان می باشد.

$$\min_{\tilde{x}} \sum_{i=1}^{L} \frac{1}{2} \left|\left|v_i - U_i\,\tilde{x}\right|\right|_2^2$$

[1] Least Square(LS)

بخش ۲ : بهینه سازی به روش توزیع شده

در بهینه سازی به صورت توزیع شده که شامل شبکه ای از عاملهای متصل به یکدیگر (نیاز به periodically strongly connected دارد) است، هدف بهینه کردن مجموعی از توابع هدف محلی بوسیله ی یک متغیر تصمیم گیری مشترک می باشد. اطلاعات صرفا بین همسایه‌های هر عامل قابل تعامل می باشد. برای انجام این پروژه از روش Alternating direction method of multipliers یا ADMM استفاده می گردد. در این روش از محاسبات تکراری استفاده می گردد. این روش از محاسبات تکراری برای هر عامل و تبادل اطلاعات بین آنها استفاده می کند. مزیت این متد در سرعت همگرایی آن است.

فرض بر این است که L عامل داریم که قرار است با همکاری هم به حل رابطه ی زیر برسند:

$$\min_{\tilde{x}} \sum_{i=1}^{L} f_i(\tilde{x}) \qquad (1)$$

الگوریتم ADMM به صورت توزیع‌شده

همانطور که گفته شده بود، شبکه ای شامل L عامل که گراف آن دارای ارتباط متقابل دوطرفه است وE تعداد یال‌های شبکه می‌باشد (به دلیل دوطرفه بودن ارتباط همانند داشتن 2E یال در گراف جهت دار می‌باشد). $G_d = \{V, A\}$ نمادی از گراف جهت دار آن می باشد و $G_d = \{V, E\}$ نماد گراف غیرجهت دار می‌باشد که V نشان دهنده ی تعداد راس‌ها و $|V| = L$ است و همچنین $|A| = E, |E| = \mathrm{E}$ است.

به صورت کلی ADMM بر روی مسله‌ی بهینه‌سازی محدب به صورت زیر قابل اجرا خواهد بود.

$$\min_{y_1, y_2} g_1(y_1) + g_2(y_2) \qquad (2)$$

$$s.t.\ C_1 y_1 + C_2 y_2 = b$$

در اینجا y_1, y_2 متغیرهای بهینه سازی می باشند و g_1, g_2 توابع بهینه سازی ما هستند. همچنین $C_1 y_1 + C_2 y_2 = b$ شرط خطی از y_1, y_2 می باشند.ADMM دنباله‌ای از زیرمسله‌ها که شامل g_1, g_2 می‌شود را یکی یکی حل می‌کند و تا زمانی که نقطه‌ی زینی وجود داشته باشد، تکرار می‌شود تا به همگرایی برسد.

برای حل مسئله (1) به روش ADMM توزیع‌شده می توان آن را به شکل زیر دوباره فرمول بندی کرد:

$$\min_{x_i, z_i} \sum_{i=1}^{L} f_i(x_i) \qquad (3)$$

$$s.t. x_i = z_{ij}\ , x_j = z_{ij}\ \forall (i,j) \in A$$

که در اینجا x_iها همان نمونه‌های محلی از متغیر $\tilde{x}$ در هر عامل i می‌باشد و z_{ij} به عنوان یک متغیر کمکی برای اجماع متغیرهای عامل‌های مختلف می‌باشد، در واقع این متغییر برای تحمیل شرط اجماع در عامل‌های همسایه i و j می‌باشد. در

شرط‌های ذکر شده، $\{x_i\}$ جدایی‌پذیرند هنگامی که $\{z_{ij}\}$ مقدار ثابت داشته باشند و بالعکس. بنابراین مسله‌ی ذکر شده در۲ می‌تواند در چارچوب محاسبات توزیع‌شده‌ی ADMM در بیاید. اگر شبکه به صورت متصل[۲] باشد، معادله‌های ۱ و ۳ به نظر یکسان می‌آیند.

حال اگر بردار $x \in R^{LN}$ برداری که x_i ها را به هم متصل کرده‌باشد و $z \in R^{2EN}$ برداری از z_{ij} ها را به هم متصل کرده‌باشد و همچنین $f(x) = \sum_{i=1}^{L} f_i(x_i)$، می‌توان رابطه ی (3) را به صورت زیر بازسازی کرد

$$\min_{x,z} f(x) + g(z) \qquad (4)$$

$$s.t.\ Ax + Bz = 0$$

که در این رابطه $g(z) = 0$ است. که در اینجا g وf توابع محدب هستند. همچنین $Ax + Bz = 0$ تابع خطی نسبت به متغیرهای بهینه سازی می‌باشد. مسئله ی ADMM، زیرمسله‌ها را که شامل f وg هستند را در یک زمان حل می کند.

در اینجا $A = [A_1; A_2]$ می‌باشد، که $A_1, A_2 \in R^{2EN \times LN}$ هر دو ترکیبی از $2E \times L$ بلوک شامل ماتریسهای $N \times N$ می‌باشد. اگر $(i,j) \in A$ باشد و z_{ij} همان q امین بلوک از z باشد، در این صورت بلوک (q,i) ام از ماتریس A_1 و بلوک (q,j) ام از ماتریس A_2 ماتریسهای I_N می باشند. در غیر این صورت، بلوکهای ما، ماتریس $N \times N$ صفر می باشند. همچنین $B = [-I_{2EN}; -I_{2EN}]$ می باشد که I ماتریس همانی است.

الگوریتم مورد استفاده

حال رابطه‌ی (4) را با اعمال ADMM حل می‌نماییم. تابع لاگرانژین به صورت زیر می‌باشد:

$$L_c(x,z,\lambda) = f(x) + \lambda\,(Ax + Bz) + \frac{c}{2}\left||Ax + Bz|\right|_2^2 \quad (5)$$

که در اینجا، λ ضریب لاگرانژ می‌باشد و c پارامتر مثبتی است. در تکرار k+1 ام، روش ADMM ابتدا تابع $L_c(x, z^{k+1}, \lambda^k)$ را بهینه نموده تا بتوان x^{k+1} را بدست آوریم. سپس $L_c(x^{k+1}, z, \lambda^k)$ را بهینه می نماییم تا z^{k+1} بدست آید. در نهایت λ^{k+1} را از x^{k+1} و z^{k+1} بروز رسانی می نماییم. روند به روز رسانی به صورت زیر می باشد:

$$\begin{aligned}
x\text{-update}: \quad & \nabla f(x^{k+1}) + A^T\lambda^k \\
& + cA^T(Ax^{k+1} + Bz^k) = 0, \\
z\text{-update}: \quad & B^T\lambda^k + cB^T(Ax^{k+1} + Bz^{k+1}) = 0, \\
\lambda\text{-update}: \quad & \lambda^{k+1} - \lambda^k - c(Ax^{k+1} + Bz^{k+1}) = 0,
\end{aligned}$$

که در صورت مشتق‌پذیر بودن تابع $f(x)$، رابطه $\nabla f(x^{k+1})$ گرادیان آن در نقطه‌ی $x = x^{k+1}$ می‌باشد و می‌تواند زیرگرادیانی از تابع غیر مشتق پذیر $f(x)$ می باشد.

[۲] connected

حال در صورتی که مقادیر اولیه‌ی z و λ به درستی انتخاب شوند، می‌توان روند به‌روزرسانی را ساده‌تر نمود. با ضرب دو طرف رابطه به‌روزرسانی متغییر λ در A^T و اضافه کردن آن به رابطه به‌روزرسانی x، رابطه‌ی زیر بدست خواهد آمد:

$$\nabla f(x^{k+1}) + A^T\lambda^{k+1} + cA^TB(z^k - z^{k+1}) = 0$$

همچنین با ضرب دو طرف رابطه به‌روزرسانی λ در B^T و اضافه کردن آن به رابطه به‌روزرسانی z، رابطه‌ی

$$B^T\lambda^{k+1} = 0$$

را خواهیم داشت. در نتیجه روابط به شکل زیر بازنویسی می گردد:

$$\nabla f(x^{k+1}) + A^T\lambda^{k+1} + cA^TB(z^k - z^{k+1}) = 0$$
$$B^T\lambda^{k+1} = 0 \qquad (6)$$
$$\lambda^{k+1} - \lambda^k - c(Ax^{k+1} + Bz^{k+1}) = 0$$

که در این رابطه $\lambda = [\beta;\gamma]$ می‌باشد و $\beta,\gamma \in R^{2EN}$ است و همچنین از رابطه‌ی دوم فرمول ۶ می‌دانیم $\beta^{k+1} = -\gamma^{k+1}$ می‌باشد. بنابراین رابطه ی اول معادله ی ۶، به صورت زیر خواهدشد:

$$\nabla f(x^{k+1}) + M_-\beta^{k+1} + cM_+(z^k - z^{k+1}) = 0$$

که در این رابطه، $M_- = A_1^T - A_2^T$ و $M_+ = A_1^T + A_2^T$. رابطه ی سوم معادله ی ۶، به دو معادله زیر قابل تقسیم است

$$\beta^{k+1} - \beta^k - cA_1x^{k+1} + cz^{k+1} = 0$$
$$\gamma^{k+1} - \gamma^k - cA_2x^{k+1} + cz^{k+1} = 0$$

اگر مقادیر اولیه λ را به صورت $\beta^0 = -\gamma^0$ انتخاب نماییم که $\beta^k = -\gamma^k, k = 0,1, \dots$ با کم و اضافه کردن دو معادله خواهیم داشت:

$$\beta^{k+1} - \beta^k - \frac{c}{2}M_-^Tx^{k+1} = 0$$
$$\frac{1}{2}M_+^Tx^{k+1} - z^{k+1} = 0 \qquad (7)$$

و همچنین اگر $z^0 = \frac{1}{2}M_+^Tx^0$ در این صورت رابطه ی 7 برای همه k ها برقرار خواهدبود

به صورت خلاصه با مقادیر اولیه‌ی z و β گفته شده خواهیم داشت:

$$\nabla f(x^{k+1}) + M_-\beta^{k+1} + cM_+(z^k - z^{k+1}) = 0$$
$$\beta^{k+1} - \beta^k - \frac{c}{2}M_-^Tx^{k+1} = 0 \quad (8)$$
$$\frac{1}{2}M_+^Tx^{k+1} - z^{k+1} = 0$$

درواقع معادله‌ی (8) الگوریتم توزیع شده‌ای می‌باشد که شامل به روز رسانی x و یک ضریب می باشد.

با جایگزینی معادله‌ی سوم رابطه‌ی ۸ در معادله ی اول آن داریم:

$$\nabla f(x^{k+1}) + M_{-}\beta^{k+1} + \frac{c}{2}M_{+}M_{+}^{T}x^{k} + \frac{c}{2}M_{+}M_{+}^{T}x^{k+1} = 0 \quad (9)$$

$$\beta^{k+1} - \beta^{k} - \frac{c}{2}M_{-}^{T}x^{k+1} = 0$$

که این رابطه از z مستقل می باشد. همچنین در معادله ی اول رابطه‌ی ۹، به‌روزرسانی x صرفا به $M_{-}\beta^{k+1}$ بستگی دارد و به β^{k} وابسته نیست. با ضرب M_{-} در معادله‌ی دوم رابطه‌ی ۹ داریم:

$$M_{-}\beta^{k+1} - M_{-}\beta^{k} - \frac{c}{2}M_{-}M_{-}^{T}x^{k+1} = 0$$

با جایگذاری آن در معادله ی اول رابطه‌ی ۹ داریم:

$$\nabla f(x^{k+1}) + M_{-}\beta^{k} + \left(\frac{c}{2}M_{+}M_{+}^{T} + \frac{c}{2}M_{-}M_{-}^{T}\right)x^{k+1} - \frac{c}{2}M_{+}M_{+}^{T}x^{k} = 0$$

حال با تعریف $W \in R^{LN\times LN}$ به عنوان بلوکی از ماتریس قطری که در آن مولفه‌ی (i,i) بلوک درجه‌ی عامل i را نشان می‌دهد که در I_N ضرب شده است و بقیه بلوک مقدار 0_N را دارد. داریم:

$$L_{+} = \frac{1}{2}M_{+}M_{+}^{T}\ , L = \frac{1}{2}M_{-}M_{-}^{T}$$

$$W = \frac{1}{2}(L_{+} + L_{-})$$

$M_{+}, M_{-}, L_{+}, L_{-}, W$ تعریف شده، به زیر ساخت توپولوژی شبکه مربوط می‌باشند. و ماتریسهای L_{+}, L_{-} ماتریس‌های لاپلاسین گسترش‌یافته[۳] بی‌علامت و علامت‌دار می‌باشند. همچنین M_{+}, M_{-} ماتریس‌های وقوع جهت‌دار و بی‌جهت[۴] گسترش‌یافته هستند. که منظور ازگسترش‌یافته جایگذاری ۱ با I_N و -۱ با $-I_N$ و 0 با 0_N در تعریف‌های اصلی این ماتریس‌ها می‌باشد.

با تعریف متغیر جدید $\alpha = M_{-}\beta \in R^{LN}$ به معادله ی ساده شده ی زیر خواهیم رسید:

$$\begin{aligned} x\text{-update}: \quad & \nabla f(x^{k+1}) + \alpha^{k} + 2cWx^{k+1} \\ & -cL_{+}x^{k} \quad = 0, \\ \alpha\text{-update}: \quad & \alpha^{k+1} - \alpha^{k} - cL_{-}x^{k+1} \quad = 0. \end{aligned}$$

به‌روزرسانی‌های فوق برای هر عامل به صورت جداگانه و توزیع شده می‌باشد. و توجه داریم $x = [x_1; \dots x_L]$ که هر x_i پاسخ محلی عامل i می‌باشد. و همچنین $\alpha = [\alpha_1; \dots \alpha_L]$ می‌باشد، که $\alpha_i \in R^N$ ضرایب لاگرانژ محلی عامل i می‌باشند.

[۳] extended

[۴] oriented and unoriented incidence matrices

با توجه به تعریف L_+, L_-, W می‌توان به‌روزرسانی‌ها را به صورت زیر بازنویسی کرد:

$$x_i^{k+1} = (\nabla f_i + 2c|\mathcal{N}_i|I)^{-1} \left(c|\mathcal{N}_i|x_i^k + c\sum_{j\in\mathcal{N}_i} x_j^k - \alpha_i^k \right)$$

$$\alpha_i^{k+1} = \alpha_i^k + c\left(|\mathcal{N}_i|x_i^{k+1} - \sum_{j\in\mathcal{N}_i} x_j^{k+1} \right)$$

که N_i مجموعه همسایه‌های عامل i می باشد. از آنجایی که مقادیر α_i و x_i به اطلاعات محلی و همسایه‌های هر عامل وابسته است، الگوریتم به صورت کاملا توزیع‌شده خواهدبود.

الگوریتم توزیع‌شده بر اساس ADMM در جدول۱ آمده‌است.

جدول۱ :الگوریتم توزیع‌شده بر اساس ADMM

Input functions f_i; initialize variables $x_i^0 = 0$, $\alpha_i^0 = 0$; set algorithm parameter $c > 0$;
For $k = 0, 1, \cdots$, every agent i do

Update x_i^{k+1} by solving $\nabla f_i(x_i^{k+1}) + \alpha_i^k + 2c|\mathcal{N}_i|x_i^{k+1} - c\left(|\mathcal{N}_i|x_i^k + \sum_{j\in\mathcal{N}_i} x_j^k \right) = 0$;

Update $\alpha_i^{k+1} = \alpha_i^k + c\left(|\mathcal{N}_i|x_i^{k+1} - \sum_{j\in\mathcal{N}_i} x_j^{k+1} \right)$;

End for

بیان مسئله به شکل توزیع‌شده

که در اینجا $\tilde{x} \in R^3$ سیگنال نامشخصی می‌باشد که می‌خواهیم آن را تخمین بزنیم و مقدار واقعی آن دارای توزیع نرمال $N(0,1)$ می‌باشد. $U_i \in R^{3\times3}$ ماتریس اندازه‌گیری[۵]خطی از عامل i می‌باشد که هر المان آن دارای توزیع نرمال است و همچنین $v_i \in R^3$ بردار اندازه‌گیری[۶] عامل i می باشدکه هر المان آن با نویز رندم $N(0,0.1)$ جمع شده است.

$$\min_{\{x_i\},\{z_{ij}\}} \sum_{i=1}^{L} \frac{1}{2} \left|\left|v_i - U_i\, x_i\right|\right|_2^2$$

$$s.t. x_i = z_{ij}, x_j = z_{ij}, \forall (i,j) \in A$$

برای حل مسله به صورت توزیع شده، معادله اصلی را به فرم معادله ۳ بازنویسی می‌کنیم. جواب بهینه با x_i^* برای هر عمل i می‌باشد. الگوریتم زمانی متوقف می گردد که تعداد تکرارها به عدد 1000 برسد و یا $\left|\left|x^k - x^*\right|\right|_2 < 10^{-15}$

پیاده‌سازی الگوریتم توزیع‌شده بر اساسADMM

ما برای پیاده سازی الگوریتم فوق در متلب ابتدا در نظر گرفتیم متغییرها را به ترتیب زیر تعریف کرده‌ایم:

متغییر بهینه‌سازی $x_i \in R^3$ ، تعداد عامل‌ها برابر با $L = 50$ و نویز جمع‌شونده برابر با$n_j = 10^{-3}$ و بردار مشاهدات $v_j \in R^3$خواهد بود.

برای تشکیل ماتریس‌ها و بردارهای مورد نظر از تابع Experiment generation استفاده کردیم که در شکل ۱ قابل مشاهده می‌باشد.

```
function [x,observationVec,measurementMat] = experimetGeneraton(numNode,numVariable,numObservation,noiseVariance)
x = randn(numVariable,1);
observationVec = zeros(numObservation,numNode);
measurementMat = randn(numObservation,numVariable,numNode);
AA = zeros(numObservation*numNode,numVariable);
BB = zeros(numObservation*numNode,1);
for ii = 1:numNode
    observationVec(:,ii) = measurementMat(:,:,ii)*x + sqrt(noiseVariance)*randn(numObservation,1);
    AA((ii-1)*3 + 1 : 3*ii,:) = measurementMat(:,:,ii);
    BB((ii-1)*3 + 1 : 3*ii,:) = observationVec(:,ii);
end
```

شکل ۱:تابع Experiment Generation

این تابع تعداد نودها و همچنین تعداد متغیرهای بهینه‌سازی هر عامل و نویز و همچنین تعداد مشاهدات را در ورودی می‌گیرد و ماتریس Uو ν را برمی‌گرداند، که در واقع observation vector همان ν میباشد و Measurement matrix همان U است و ما در هر ازمایش یک این بردارها را به صورت تصادفی می‌سازیم و در نهایت برای حذف تاثیر تصادفی بودن متغییرها پاسخ مسله را روی تعداد ۱۰ آزمایش متوسط می‌گیریم و پاسخ متوسط گرفته شده را رسم می‌کنیم.

[۵] Measurement matrix

[۶] Measurement vector

در ادامه می‌خوایم که عامل‌ها را در شبکه‌های متفاوت قرار دهیم و تاثیر شبکه‌های مختلف اعم از گراف کامل، گراف ستاره و... را بر پاسخ نهایی و سرعت همگرا شدن ببینیم. به این منظور از تابع NetworkGeneration که طریقه‌ی پیاده سازی آن در شکل ۲ آمده است استفاده میکنیم.

```
function [adjacencyMat] = networkGeneration(treeFalg,branchFactor,regularFlag,degree,numNode,numEdge)
if treeFalg==1
    edgeList = canonicalNets(numNode,'tree',branchFactor);  % Constructing edge lists for simple canonical graphs, ex: trees and lattices.
    adjacencyMat = edgeL2adj(edgeList);
elseif regularFlag==1
    edgeList = kregular(numNode,degree);    % Create a k-regular graph.
    adjacencyMat = edgeL2adj(edgeList);
else
    connectedFlag = 0;
    while connectedFlag==0
        adjacencyMat = randomGraph(numNode,.5,numEdge);    % Random graph construction routine.
        connectedFlag = isConnected(adjacencyMat);
    end
end
```

شکل ۲ : تابع Network generation

در توضیح متغییرهای مختلف تابع فوق، ابتدا treeflag میباشد که در صورت یک بودن آن وارد تابع cannonicalNets خواهیم شد که با گرفتن تعداد نودهای گراف و همچنین branchfactor درخت مورد نظر را می‌سازد. مقدار branchfactor نشان دهنده‌ی تعداد یال‌های میباشد که در هنگام ساخت گراف به نود اول متصل میشود و سپس به نود بعدی، به عنوان مثال اگر branchfactor = numberofnodes-1 باشد، درخت ساخته شده به صورت گراف ستاره خواهد بود و اگر branchfactor = 1 باشد گراف خط را خواهیم داشت.

سپس از گراف k-regular استفاده کردیم برای ساخت شبکه‌هایی به صورت دایره[۷] که به این صورت می‌باشد که تابع در ورودی تعداد نودها و همچنین تعداد درجه‌ی هر نود را می‌گیرد ، و در نهایت برای ساخت گراف کامل با تعیین تعداد یال برابر تعداد گراف کامل(n(n-1))/2 میتوانیم با استفاده از تابع randomgraph به گراف کامل دست پیدا کنیم. تابع networkgeneration ماتریس مجاورت شبکه‌ی مورد نظر را برمی‌گرداند.

توابعی که برای ساخت گراف‌ها استفاده شده‌اند را از کتابخانه‌ی octave-networks-toolbox برداشتیم.

[۷] cycle

برای پیاده‌سازی الگوریتم فوق الذکر از تابع ADMM_Dist استفاده کردیم که در شکل ۳ آمده است.

```
function [errorNormalizedDist] = ADMM_Dist(adjMatrixOrig,rho,numNode,numberVariable,numIteration,A,b,xHatCen)

xHatMatrixNew = zeros(numberVariable,numNode);
xHatMatrixNew2 = zeros(numberVariable,numNode);
alphaMat = zeros(numberVariable,numNode);
errorNormalizedDist = zeros(numIteration,1);
lastRCVDMat = zeros(numberVariable,numNode,numNode);

for jj = 1:numIteration

        for ii = 1:numNode
            xHatMatrixOld = xHatMatrixNew;
            nodeInd = ii;
            nodeIndex = nodeInd;
            numNeighbor = sum(adjMatrixOrig(nodeInd,:));
            lastRCVDMat(:,:,nodeInd) = repmat(adjMatrixOrig(nodeInd,:),size(xHatMatrixNew,1),1).*xHatMatrixOld;
            alphaMat(:,nodeInd) = alphaMat(:,nodeInd) + rho*(numNeighbor*xHatMatrixOld(:,nodeInd) - sum(lastRCVDMat(:,:,nodeInd),2)) ;
            ACV = A(:,:,nodeIndex);
            bCV = b(:,nodeIndex);
            xHatMatrixNew2(:,nodeIndex) = inv(ACV'*ACV + 2*rho*numNeighbor*eye(numberVariable))*( ACV'*bCV - alphaMat(:,nodeIndex) + ...
            rho*(numNeighbor*xHatMatrixOld(:,nodeIndex) + sum(lastRCVDMat(:,:,nodeIndex),2)));
        end
        xHatMatrixNew = xHatMatrixNew2;
       % xHatMatrixIterDist(:,:,jj) = xHatMatrixNew;
    errorNormalizedDist(jj,:) = sqrt(sum(sum((xHatMatrixNew - repmat(xHatCen,1,numNode)).^2)))/sqrt(sum(sum(repmat(xHatCen,1,numNode).^2)));

end
```

شکل ۳ : تابع ADMMM_Dist برای پیاده سازی الگوریتم توزیع شده

این تابع در ورودی خود ماتریس مجاورت گراف، ضریب ADMM، تعداد عامل‌ها، تعداد دفعات تکرار الگوریتم، ماتریسU، بردار νوهمچنین مقدار بهینه مسله در حالت متمرکز را می‌گیرد، و یک بردار خطای نرمالیزه برمیگرداند، که در واقع این خطا نسبت به روش متمرکز محاسبه شده است.

با توجه به الگوریتم، باید در هر تکرار ابتدا هر عامل مقدار x خود را محاسبه می‌کند وپس از آن مقدار α در هر عامل به‌روزرسانی می‌شود، و در نهایت خطای مقدار x حاصله از مقدار جواب بهینه که در روش متمرکز حساب کرده بودیم به شکل زیر محاسبه می‌کنیم:

$$normalized\ error\ of\ iteration\ k = \frac{\sqrt{\sum_{j=1}^{L}\sum_{i=1}^{N}\left(x_{ji}^{k} - \hat{x}_{\iota}\right)^2}}{\sqrt{\sum_{j=1}^{L}\sum_{i=1}^{N}(\hat{x}_{\iota})^2}} = \frac{\sqrt{\sum_{j=1}^{L}\sum_{i=1}^{N}\left(x_{ji}^{i} - \hat{x}_{\iota}\right)^2}}{\sqrt{L \times \sum_{i=1}^{N}\hat{x}_{\iota}}}$$

که در اینجا $\hat{x}$ پاسخ حل معادله به روش مرکزی می باشد و $\hat{x}_{\iota}$ i امین متغیر $\hat{x}$ است. خطای نرمالیزه شده از روش normalized mean square error (NMSE) بدست می آید که به صورت زیر می‌باشد:

$$NMSE(x,y) = MSE(x,y)/MSE(x,0) = \frac{\| x - y \|_2^2}{||x||_2^2}$$

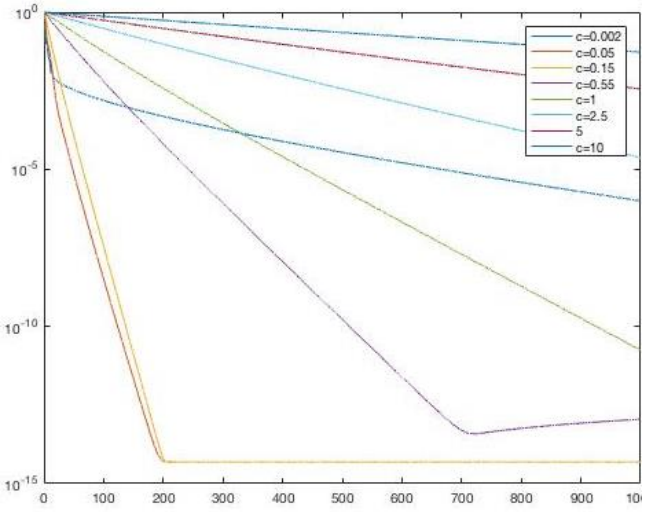
برای بدست آوردن خطای نسبی از روش NMSE استفاده کرده‌ایم که این متد، نسبت میزان انحراف معیار از خروجی واقعی را به اندازه‌ی سیگنال واقعی (انحراف معیار سیگنال واقعی از صفر) مورد بررسی قرار می دهد. به همین دلیل، NMSE به طور کلی،

تفاوت های چشمگیر بین مدل ها را نشان خواهد داد. در صورتی که مدلی دارای NMSE بسیار کم باشد، عملکرد مدل در همه حالات، مورد قبول است. از این روش بدست آوردن خطا در پردازش سیگنال به مراتب استفاده می شود.

تاثیر ضریب ADMMM بر روی همگرایی:

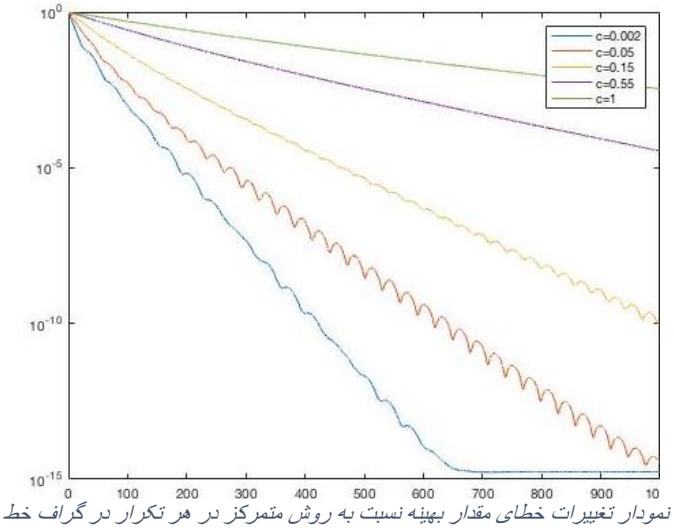
از آنجایی که ما نمیدانیم مقدار بهینه‌ی ضریب ADMM در هر روش چه مقدار می‌باشد، در هر یک از گراف‌هایی که می‌سازیم، برای این ضریب مقادیر مختلفی بین 0 تا 5 و برای بعضی مدل‌ها حتی مقدار ضریب را بالاتر می‌بریم تا اینکه ببینیم مقدار خطا از حالت متمرکز در کدام ضریب کمتر و سرعت همگرایی نیز بیشتر می‌شود.

برای گراف کامل وقتی که تعداد عامل‌ها برابر با ۵۰ بود، ضرایب ADMM را بین [10 0.002] قرار دادیم ودر شکل ۴می‌بینم که بهینه‌ترین پاسخ وقتی بدست می‌آید که $\rho = 0.05$ می‌باشد. در این حالت همگرایی بعد از ۲۰۰ تکرار حاصل می‌شود.



شکل ٤: نمودار تغییرات خطای مقدار بهینه نسبت به روش متمرکز در هر تکرار در گراف کامل

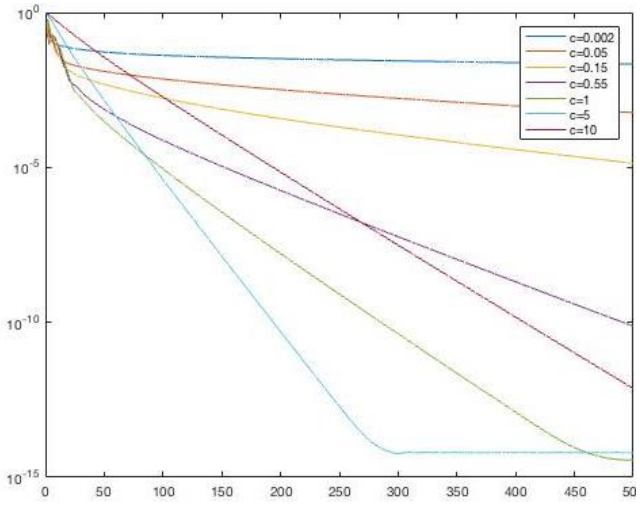
در گراف خط، به دلیل آنکه سرعت انتقال اطلاعات به شدت پایین بود، برای همگرایی ۵۰ نود زمان بسیار زیادی طول می‌کشید، لذا ما به جای ۵۰ عامل مقدار بهینه‌ی ضریب ADMM را روی ۱۵ عامل بررسی کردیم ، و در شکل ۵ میبینیم که مقدار بهینه‌ی



شکل ٥: نمودار تغییرات خطای مقدار بهینه نسبت به روش متمرکز در هر تکرار در گراف خط

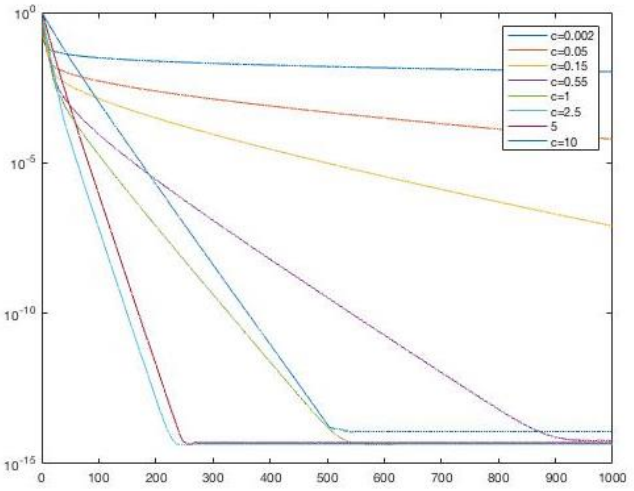
$\rho = 0.002$ خواهد بود. که در این حالت همگرایی بعد از تقریبا ۷۰۰ تکرار رخ داده است. و خطا تقریبا برابر با 10^{-15} خواهد بود.

در حالت سوم گراف ستاره را مورد بررسی قرار دادیم، که مقدار ضریب بهینه‌ی آن برابر با $\rho = 5$ خواهد بود و همچنین همگرایی نیز پس از ۳۰۰ تکرار رخ خواهد داد، نمودارهای آن در شکل ۶ قابل مشاهده می‌باشد.



شکل ٦: نمودار تغییرات خطای مقدار بهینه نسبت به روش متمرکز در هر تکرار در گراف ستاره

و در نهایت گراف دایره را بررسی کردیم که مقدار بهینه ضریب را $\rho = 2.5$ بدست‌می‌آوریم، و در شکل ۷ مشاهده می‌کنیم که همگرایی پس از ۲۰۰ تکرار رخ خواهد داد.



شکل ۷: : نمودار تغییرات خطای مقدار بهینه نسبت به روش متمرکز در هر تکرار در گراف دایره

مقادیر ضرایب ADMM بدست آمده لزوما بهترین و بهینه‌ترین مقدار در این روش نمی‌باشد و تنها ما با امتحان کردن تعدادی از مقادیر مختلف این مقدارها را بدست آوردیم.

بخش ۳: بهینه سازی به روش غیر متمرکز

در این روش ابتدا حالتی را در نظرمی‌گیریم که یک متغییر سراسری مشترک داریم، که تابع هدف و شروط به N قسمت تقسیم شده اند، و مسله به شکل زیر خواهد شد.

$$\min_{x} \quad f(x) = \sum_{i=1}^{L} f_i(x)$$

و همانطور که می‌دانیم هدف حل مسله فوق به صورتی است که هریک از f_i بتوانند به تنهایی و جداگانه مقدار بهینه‌ی خود را محاسبه کنند، لذا در حالت غیر متمرکز می‌توان مسله را به صورت زیر نوشت:

$$\min_{x_i} \sum_{i=1}^{L} f_i(x_i)$$

$$s.t. \;\; x_i = z \qquad i = 1, \dots N$$

در این متغیرهای محلی به صورت $x_i \in R^n$ می باشد و متغیر سراسری[۸] را با z تعریف می‌کنیم. اگر تمام متغییرهای محلی به اتفاق برسند(به عنوان مثال برابر شوند) می‌توان به این مسئله اجماع کلی گفت.

ADMM مسئله فوق میتواند به صورت مستقیم از لانگرانژین افزونی بدست می‌آید:

$$L_c(x_1, \dots, x_N, z, \lambda) = \sum_{i=1}^{L} f_i(x_i) + \lambda_i^T (x_i - z) + \frac{c}{2} ||x_i - z||_2^2$$

مجموعه ی شروط بدین صورت است :

$$\mathcal{C} = \{(x_1, \dots, x_N) \mid x_1 = x_2 = \cdots = x_N\}.$$

الگوریتم ADMM آن بدین صورت خواهد بود

$$x_i^{k+1} = \underset{x_i}{\operatorname{argmin}} \, f_i(x_i) + \lambda_i^T (x_i - z) + \frac{c}{2} ||x_i - z||_2^2$$

$$z^{k+1} = \sum_{i=1}^{N} x_i^{k+1} + (\frac{c}{2})\lambda_i^k$$

$$\lambda_i^{k+1} = \lambda_i^k + c(x_i^{k+1} - z^{k+1})$$

[۸] global

در این الگوریتم المان پردازشی متغیر سراسری z را مدیریت می نماید، در واقع منظور از المان پردازشی همان هماهنگ-کننده[۹]عامل ها می‌باشد. باید در نظر داشت که به روز رسانی z، با پیش افکندن $x^{k+1} + \frac{1}{\rho} y^k$ بر روی مجموعه C می باشد. در هر تکرار هر عامل مقدار x_i^{k+1} خود را محاسبه می‌کند و آن را به هماهنگ‌کننده میدهد، هماهنگ‌کننده x_i^{k+1} های مربوط به تمام عامل‌ها را می‌گیرد و مقدار z آن مرحله را محاسبه می‌کند و به عامل‌ها برمی‌گرداند و آنها λ_i^{k+1} را به روز رسانی می‌کنند.

پیاده‌سازی الگوریتم غیرمتمرکز بر اساس ADMM

مسئله حداقل مربعات در روش غیرمتمرکز به شکل زیر در می‌اید:

$$\min_{\{x_i\}} \sum_{i=1}^{L} \frac{1}{2} \left|\left| \nu_i - U_i\, x_i \right|\right|_2^2$$

$$s.t. \qquad x_i = z$$

برای پیاده سازی آن در متلب همانند حالت توزیع شده تعداد عامل‌ها و نویز را به این صورت تعریف می‌کنیم که، متغییر بهینه‌سازی $x_i \in R^3$ ، تعداد عامل ها برابر با $L = 50$ و نویز جمع‌شونده برابر با $n_j = 10^{-3}$ و بردار مشاهدات $\nu_j \in R^3$خواهد بود.

همانند شکل ۱ که الگوریتم توزیع یافته برای ساخت ماتریس‌ها و بردارها از تابع ExperimentGeneration استفاده کردیم، در اینجا نیز همان تابع را استفاده میکنیم، و همچنین از آنجایی که میخواهیم دو روش را باهم مقایسه کنیم، و ضرایب را به صورت تصادفی تعریف می‌شود، لذا در هر مرحله آزمایش که انجام میشود، هر دو تابع غیرمتمرکز و توزیع‌شده باهم محاسبه میشوند و خطاهای آنها نسبت به حالت متمرکز محاسبه میشود، و در نهایت مقدار بدست‌امده در هر آزمایش روی تمام آزمایش‌ها متوسط گرفته می‌شود.

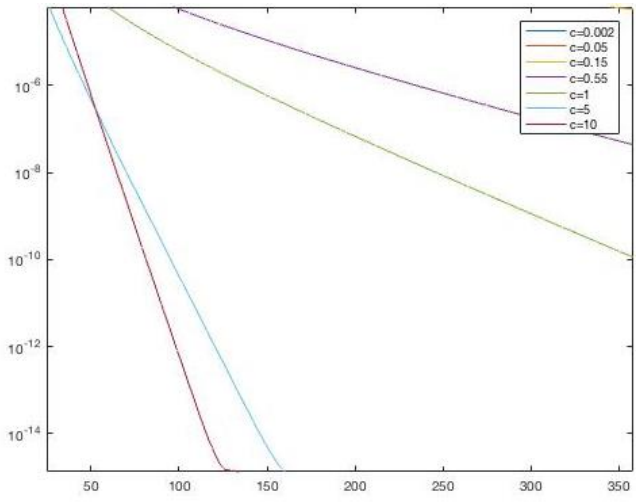
برای پیاده‌سازی الگوریتم از تابع ADMM_Decen استفاده می‌کنیم که در شکل ۸ روند پیاده‌سازی تابع قابل مشاهده است.

[۹] coordinator

```
function [ errorNormalizedDecen ] = ADMM_Decen( c,numAgent,numVariable,numIteration,measurementMat,observationVec,xHatCen)
errorNormalizedDecen = zeros(numIteration,1);
xHatMatrixNew = zeros(numVariable,numAgent);

Yold = zeros(numVariable,numAgent);
Zold = zeros(numVariable,numAgent);

for jj = 1:numIteration
        for ii = 1:numAgent
            AgentIndex = ii;
            ACV = measurementMat(:,:,AgentIndex);
            bCV = observationVec(:,AgentIndex);
            xHatMatrixNew(:,AgentIndex) = ((ACV)'*((ACV)) + c*eye(numVariable))\(c*Zold(:,AgentIndex) - Yold(:,AgentIndex) + ((ACV)')*bCV);

        end

        Znew = repmat(mean(xHatMatrixNew + (1/c)*Yold,2),1,numAgent);
        Ynew = Yold + c*(xHatMatrixNew-Znew);

        Yold = Ynew;
        Zold = Znew;

 % xHatIterDecenter(:,:,jj) = xHatMatrixNew;

    errorNormalizedDecen(jj,:) = sqrt(sum(sum((xHatMatrixNew - repmat(xHatCen,1,numAgent)).^2)))/sqrt(sum(sum(repmat(xHatCen,1,numAgent).^2)));




end
```

شکل ۸: روند پیاده‌سازی الگوریتم غیر متمرکز

این تابع نیز مانند حالت توزیع‌شده در ورودی ماتریس مجاورت گراف، ضریب ADMM، تعداد عامل‌ها، تعداد دفعات تکرار الگوریتم، ماتریسU، بردار νوهمچنین مقدار بهینه مسله در حالت متمرکز را می‌گیرد، و یک بردار خطای نرمالیزه برمیگرداند، که در واقع این خطا نسبت به روش متمرکز محاسبه شده است، آن را در یک بردار ریخته که میتوان در نهایت میزان نزدیکی پاسخ را به مقدار متمرکز نشان می‌دهد.در این روش نیز، ما مقادیر مختلف ضریب ADMM را در نظر میگیریم که ببینیم در کدام مقدار پاسخ بهتر و سرعت همگرایی نیز بالاتر خواهد بود، در شکل ۹ نمودار تغییرات ضرایب ADMM میبینم و مشاهده می‌کنیم که در این روش بهینه‌ترین ضریب مقدار$\rho = 10$ را خواهد داشت. که در تقریبا ۱۰۰ تکرار همگرا میشود.

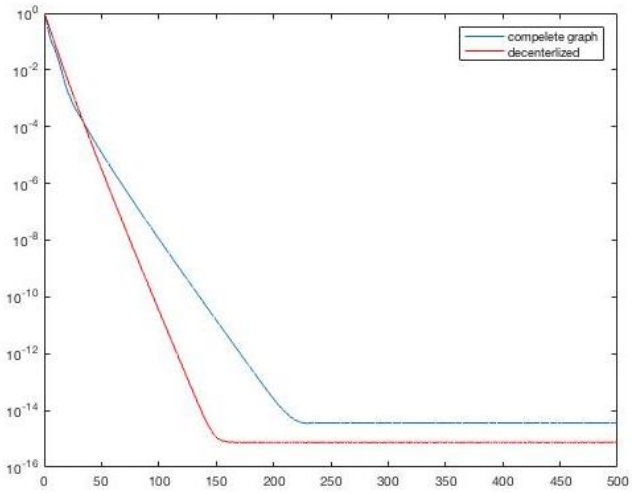


شکل ۹:ضرایب ADMM در الگوریتم غیرمتمرکز

مقایسه دو روش

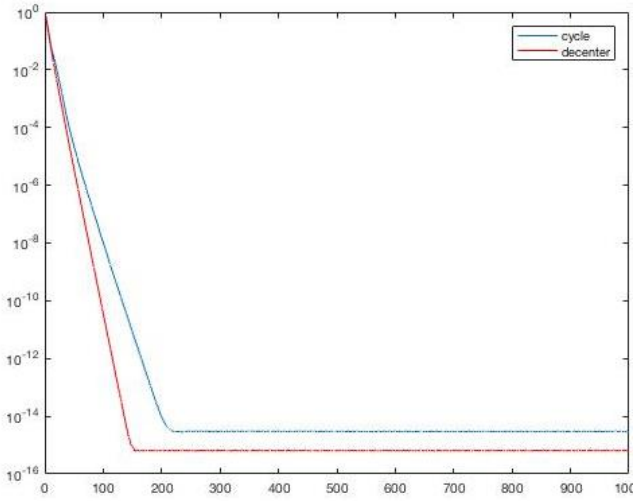
در این قسمت میخواهیم ببینیم که آیا روش توزیع شده جواب بهتری به ما می‌دهد یا روش غیرمتمرکز. منظور از جواب بهتر پاسخی است که سرعت همگرایی آن بالاتر باشد و میزان خطای آن در نهایت کمتر باشد و به پاسخ حالت متمرکز نزدیک‌تر شود.

در ابتدا گراف کامل را با روش غیرمتمرکز مقایسه کردیم که در شکل ۱۰ آمده است. همانطور که در شکل می‌بینیم، در این حالت روش غیرمتمرکز بسیار بهتر عمل‌کرده است، و ۱۰۰ مرحله زودتر به مقدار بسیار کمتری نسبت به حالت توزیع یافته همگرا شده است.



شکل ۱۰: مقایسه دو روش غیر متمرکز و توزیع‌شده با شبکه گراف کامل

در حالت دوم گراف دایره را مقایسه می‌کنیم، در شکل ۱۱ می‌بینیم که همچنان سرعت همگرایی روش غیرمتمرکز بیشتر است.



شکل ۱۱: : مقایسه دو روش غیر متمرکز و توزیع‌شده با شبکه گراف کامل

بخش۴: دیگر کاربردهای تابع حداقل مربعات

مکان‌یابی نودهای حس‌گر به دلیل تصادفی توزیع آنها بعد از استقرار، امر قابل توجهی می‌باشد. با استفاده از تابع حداقل مربعات می‌توان روندی برای مکان‌یابی آنها با در نظر گرفتن کاهش خطا بدست آورد.

در شبکه‌های حس‌گر بی‌سیم[۱۰]، تعدادی حس‌گر به صورت بی‌سیم با یکدیگر در ارتباطند. یک شبکه حس‌گر بی‌سیم در اطراف ناحیه مورد نظر و یا داخل یک شی قرار گیرد، و حس‌گرها قادر هستند تا پارامترهای مختلف محیطی را تحت‌نظر قرار دهند و آنها را به یک نود مرکزی[۱۱] منتقل نمایند. کاربردهای جدیدی که شبکه‌های حس‌گر بی‌سیم فراهم‌کرده‌است در تشخیص به موقع آتش‌سوزی جنگل و همچنین در نظارت بر دیواره‌ی سدهای مصنوعی می‌باشد.

داده‌های حاصل تنها زمانی معنی‌دار می‌شود که با موقعیت جغرافیایی حس‌گرها ترکیب گردد.

تخمین موقعیت نقطه‌ی نامشخص P(x,y) نیازمند حداقل سه نقطه شناخته در دو بعد می باشد. با داشتن m موقعیت جغرافیایی $B(x_i, y_i)$ و فاصله‌ی r_i از آنها داریم:

$$(x - x_i)^2 + (y - y_i)^2 = r_i^2 \, , i = 1, \dots, m$$

این عبارت می بایست با ابزارهای خطی‌سازی از جمله سری تیلور خطی گردد.

با اضافه و کم کردن y_j, x_j به همه معادلات داریم:

$$\left(x - x_j + x_j - x_i\right)^2 + \left(y - y_j + y_j - y_i\right)^2 = r_i^2 \, , i = 1, \dots, m, j = 1, \dots, m$$

با داشتن فاصله ی r_i و یا r_j که فاصله ی نقطه ی نامعلوم تا نود مرکزی شماره ی i یا j می‌باشد و فاصله‌ی d_{ij} که فاصله‌ی دو نود مرکزی B_j و B_i از هم می باشند، با ساده سازی داریم:

$$(x - x_j)(x_i - x_j) + (y - y_i)(y_i - y_j) = \frac{1}{2}\left[r_j^2 - r_i^2 + d_{ij}^2\right] = b_{ij}$$

بدلیل حائز اهمیت نبودن اینکه کدام معادله به عنوان ابزرای برای خطی سازی استفاده گردد، j=1 برای ما کفایت می کند که بدین معناست که اولین نود مرکزی انتخاب گردد و منجر به ایجاد سیستم خطی با m-1 معادله و n=2 مجهول داریم.

$$\begin{aligned}
(x - x_1)(x_2 - x_1) + (y - y_1)(y_2 - y_1) &= \tfrac{1}{2}\left[r_1^2 - r_2^2 + d_{21}^2\right] = b_{21} \\
(x - x_1)(x_3 - x_1) + (y - y_1)(y_3 - y_1) &= \tfrac{1}{2}\left[r_1^2 - r_3^2 + d_{31}^2\right] = b_{31} \\
&\vdots \\
(x - x_1)(x_m - x_1) + (y - y_1)(y_m - y_1) &= \tfrac{1}{2}\left[r_1^2 - r_m^2 + d_{m1}^2\right] = b_{m1}
\end{aligned}$$

[۱۰] Wireless sensor networks(WSN)

[۱۱] beacon

که می توان این معادله را به فرم ماتریسی نوشت:

$$Ax = b$$

$$A = \begin{pmatrix} x_2 - x_1 & y_2 - y_1 \\ x_3 - x_1 & y_3 - y_1 \\ \vdots & \vdots \\ x_m - x_1 & y_m - y_1 \end{pmatrix}, \mathbf{x} = \begin{pmatrix} x - x_1 \\ y - y_1 \end{pmatrix}, \mathbf{b} = \begin{pmatrix} b_{21} \\ b_{31} \\ \vdots \\ b_{m1} \end{pmatrix}.$$

با توجه به این حقیقت که سیستم‌های معادلات بیش‌از حد تعیین شده با $m \gg n$ راه‌حل واحد دقیقی برای $Ax = b$ ندارند، ما نرم دوم(نرم اقلیدسی) که مجموع مربعات را مینیمم می‌کند را اعمال می‌کنیم. که به صورت معادله ی زیر می شود:

$$\min_x ||Ax - b||_2^2 = \sum_{i=1}^{n} \frac{1}{2} ||b_i - a_i\, x_i||_2^2$$

حل این مسئله با روش ADMM که در بخش قبلی بیان شده، می باشد. در اینجا x, y دو موقعیت جغرافیایی را بیان می نمایند.

بخش ۵: منابع

[1] Shi, W., Ling, Q., Yuan, K., Wu, G. and Yin, W., 2014. On the Linear Convergence of the ADMM in Decentralized Consensus Optimization. IEEE Trans. Signal Processing, 62(7), pp.1750-1761.

[2] Boyd, S., Parikh, N., Chu, E., Peleato, B. and Eckstein, J., 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. Foundations and Trends® in Machine learning, 3(1), pp.1-122.

[3] Sherif Abdelwahab , Mohamed Grissa , Nitin Jogee Bella Subramanian Solving Consensus Least Squares in Networking with the Alternating Direction Method of Multipliers

[4] Reichenbach, F., Born, A., Timmermann, D. and Bill, R., 2006, June. A distributed linear least squares method for precise localization with low complexity in wireless sensor networks. In International Conference on Distributed Computing in Sensor Systems (pp. 514-528). Springer, Berlin, Heidelberg.