Studio di un algoritmo di consensus Newton-Raphson per l'ottimizzazione distribuita di funzioni convesse

Andrea Bernardi 626356 - Enrico Regolin 626358 - Giulio Veronesi 1014400
Laurea Magistrale in Ingegneria dell'Automazione
Progettazione di Sistemi di Controllo A.A 2011-2012

26/02/2012

***Abstract*—Un problema di recente interesse nelle reti distribuite è la progettazione di algoritmi per la minimizzazione di somme di funzioni costo convesse, dove ogni funzione è nota solo a ciascun agente che compone la rete.**
Questi agenti devono cooperare con lo scopo di trovare il minimo della somma di tutti i costi.
Algoritmi di questo tipo trovano interessanti applicazioni in ambiti nei quali non è possibile o economico avere un controllo centralizzato della rete che abbia accesso a tutte le informazioni dei singoli agenti che la compongono. Esempi di applicazioni riguardano l'allocazione ottimale di risorse, il controllo del traffico aereo e stradale e la stima in linea di parametri da una rete di sensori (problemi di regressione lineare e robusta).

Questo elaborato nasce come approfondimento del lavoro documentato in [1] nel quale si è affrontato il problema dell'ottimizzazione convessa distribuita introducendo un algoritmo di Newton-Raphson per la ricerca del minimo di una funzione regolare e sfruttando il principio della separazione di scale temporali per il raggiungimento del *consensus* tra gli agenti, come descritto in [13].
In questo lavoro si illustra un'applicazione di questo algoritmo ad un problema di regressione distribuita, sia nel caso lineare sia nel caso di regressione robusta.

Importante ruolo gioca il parametro ε nel regolare le due scale temporali: verrà ampiamente studiato il suo comportamento e varranno proposte delle strategie per il suo *tuning* ottimale.

***Index Terms*—distributed optimization, distributed regression, convex optimization, consensus algorithms, Newton-Raphson method, least square and robust regression, gossip algorithm.**

I. Introduzione

Nell'ultimo decennio gli sviluppi dei sistemi di comunicazione e la riduzione dei costi di sensori e unità di calcolo hanno aperto la strada a molte innovazioni nel campo dell'ingegneria dell'informazione. In particolare la progettazione di sistemi di controllo si è spostata spesso dai tradizionali approcci centralizzati, dove tutte le informazioni delle rete sono concentrate in pochi nodi e le decisioni sono assunte da un livello superiore di controllo, ad approcci distribuiti, dove cioè ogni agente possiede e calcola autonomamente un'informazione locale ma le decisioni vengono prese cooperativamente con gli altri agenti attraverso lo scambio di alcune informazioni senza che vi sia una supervisione superiore.

In teoria dei sistemi con il termine *consensus* si intende la convergenza di più agenti ad un parametro comune mediante scambio di informazioni tra di essi o interazione locale.
In molte applicazioni, inoltre, si desidera non solo il raggiungimento del *consensus* tra gli agenti della rete, ma anche che l'informazione associata sia ottima.

L'algoritmo in esame si basa sulla minimizzazione di una funzione di costo globale convessa data dalla somma di funzioni costo convesse disponibili "privatamente" a ciascun agente che compone la rete.
Ci sono grandi vantaggi nel formulare un problema come problema di ottimizzazione convessa, anzitutto può essere positivamente ed efficientemente risolto, talvolta anche in modo immediato come nel caso dei minimi quadrati, inoltre ben si adatta a numerosi casi di nostro interesse: circuiti elettronici, analisi di dati, statistica, finanza, controlli automatici etc.

Gli algoritmi di ottimizzazione distribuita si possono suddividere in tre grandi categorie: metodi di decomposizione primale, duale e metodi euristici.
L'ultima categoria comprende algoritmi ricavati per via euristica che vengono adattati al problema in esame, per questo motivo in generale non è possibile farne un preciso studio analitico.
Le prime due categorie invece sono caratterizzate dal fatto che il problema originario di ottimizzazione viene suddiviso in vari sottoproblemi di solito indipendenti e più semplici da risolvere; nel caso dei metodi di decomposizione primale si lavora esclusivamente con le variabili del sistema originale, nell'altro caso si operano ottimizzazioni anche nel sistema duale attraverso il metodo dei moltiplicatori di Lagrange.
I metodi di decomposizione primale possono essere interpretati come un'ottimizzazione nella quale globalmente vengono fornite delle risorse ad ogni sottoproblema, gli agenti operano per ottenere il massimo possibile con le risorse a disposizione ed i risultati vengono rielaborati, riassegnando nell'iterazione successiva altre risorse agli agenti per migliorare il risultato precedente fino, asintoticamente, a raggiungere il valore di ottimo.
La decomposizione duale può essere interpretata invece secondo una visione economica: globalmente viene imposta dal sistema duale una funzione prezzo alle risorse disponibili, ogni

agente lavora indipendentemente con l'obiettivo di minimizzare il proprio costo a seconda della funzione prezzo imposta e di nuovo, all'iterazione successiva, si aggiusta questa funzione prezzo con lo scopo di ottenere la migliore strategia domanda/offerta.

L'algoritmo oggetto della relazione appartiene alla categoria degli algoritmi di decomposizione primale: essi sono facili da implementare, ampiamente utilizzabili, richiedono poche ipotesi, tra le quali è fondamentale quella della convessità delle funzioni costo, e non è necessaria la sincronizzazione degli agenti. Sostanzialmente questo algoritmo converge al valore di minimo della funzione costo globale attraverso il metodo di Newton-Raphson, sufficientemente lento da permettere il raggiungimento del *consensus* tra gli agenti. Ciò che consente il funzionamento dell'algoritmo è il principio di separazione di scale temporali, descritto nel testo di Khalil [13], nello specifico il parametro ε che verrà ampiamente esaminato nel seguito della relazione.

Gli algoritmi di *consensus* possiedono inoltre numerosi vantaggi pratici: robustezza ai rumori di misura, robustezza ai cambiamenti della topologia della rete, richiedono pochi sforzi computazionali e garantiscono la convergenza per il problema globale. Esistono naturalmente dei punti critici, per esempio la forte influenza che esercitano gli *outliers* negli schemi più semplici, quelli che adottano la minimizzazione ai minimi quadrati.
In questa relazione si proporrà allo scopo una soluzione alternativa che adotta una funzione di costo "robusta".

II. Descrizione dell'algoritmo

Si ritiene utile dare delle definizioni preliminari di *consensus* e *average consensus*:

Consideriamo il sistema lineare, dinamico, a tempo discreto:

$$\mathbf{x}(k+1) = P(k)\mathbf{x}(k) \tag{1}$$

dove $\mathbf{x}(k) = [x_1(t)x_2(t)\dots x_N(k)]^T \in \mathbb{R}^N$ e $P(k)$ è stocastica o doppiamente stocastica. L'equazione (1) può essere scritta nella forma:

$$x_i(k+1) = x_i(k) + \sum_{j=1}^{N} p_{ij}(k)(x_j(k) - x_i(k)) \qquad i = 1, \dots, N$$

che mette in evidenza l'aggiornamento di ogni componente del sistema. Si dice che $P(k)$ risolve il problema di *consensus* se $x_i(k) \xrightarrow{k\to+\infty} \alpha \ \forall i$. $P(k)$ risolve il problema di *average consensus* se, oltre alla precedente condizione, si ha la convergenza nella media degli stati iniziali $\alpha = \frac{1}{N}\sum_{i=1}^{N} x_i(0)$. Se $P(k) = P$ si ha il caso di matrice di *consensus* tempo-invariante.

A. Formulazione del problema

In [1] viene proposto un algoritmo distribuito per il calcolo della soluzione ottima della funzione costo globale:

$$\overline{f} : \mathbb{R} \to \mathbb{R} \qquad \overline{f}(x) := \frac{1}{N}\sum_{i=1}^{N} f_i(x) \tag{2}$$

dove si è supposto che ciascun agente abbia lo scopo di minimizzare in maniera distribuita la funzione di costo globale $\overline{f}$. Si ipotizza inoltre che le funzioni di costo "private" siano $f_i : \mathbb{R} \to \mathbb{R} \in C^2 \ \forall i$, con derivate seconde strettamente positive, limitate e definite per ogni $x \in \mathbb{R}$. L'obiettivo di ogni agente è quello di minimizzare $\overline{f}$ attraverso l'algoritmo di ottimizzazione distribuita, che deve quindi raggiungere asintoticamente il valore di ottimo:

$$x^* = \operatorname*{argmin}_x \overline{f}(x) \tag{3}$$

Si esprime la topologia della rete tramite un grafo non orientato e connesso $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, dove $\mathcal{V}$ sono gli agenti ed $\mathcal{E}$ sono le connessioni tra essi.
Consideriamo uno schema di comunicazione sincrono ed una rete di comunicazioni strutturata in modo che la matrice di *consensus* P associata al grafo sia positiva e doppiamente stocastica, cioè vale:

$$P\mathbb{1} = \mathbb{1} \qquad \mathbb{1}^T P = \mathbb{1}^T$$

Come noto, questo equivale ad imporre la convergenza di tutti gli agenti nel baricentro dello stato iniziale, ovvero nella media:

$$\overline{x} = \frac{1}{N}\sum_{i=1}^{N} x_i(0) \tag{4}$$

Vengono introdotte le seguenti notazioni:

$$g_i(x_i(k)) := f_i''(x_i(k)) \cdot x_i(k) - f_i'(x_i(k)) \tag{5}$$

$$h_i(x_i(k)) := f_i''(x_i(k))$$

$$\mathbf{x}(k) := \begin{bmatrix} x_i(k) \\ \vdots \\ x_N(k) \end{bmatrix} \tag{6}$$

$$\mathbf{g}(\mathbf{x}(k)) := \begin{bmatrix} g_1(x_i(k)) \\ \vdots \\ g_1(x_N(k)) \end{bmatrix} \tag{7}$$

$$\mathbf{h}(\mathbf{x}(k)) := \begin{bmatrix} h_1(x_i(k)) \\ \vdots \\ h_1(x_N(k)) \end{bmatrix} \tag{8}$$

Infine si utilizzerà la barra di frazione per indicare la divisione componente per componente tra due vettori:

$$\frac{\mathbf{g}(\mathbf{x}(k))}{\mathbf{h}(\mathbf{x}(k))} = \begin{bmatrix} \frac{g_1(x_1(k))}{h_1(x_1(k))} & \dots & \frac{g_1(x_N(k))}{h_1(x_N(k))} \end{bmatrix}^T \tag{9}$$

B. Distribuited Newton-Raphson consensus

L'algoritmo sfrutta il fatto che il minimo globale di una somma di funzioni costo quadratiche del tipo:

$$f_i(x) = \frac{1}{2}a_i(x - b_i)^2 \tag{10}$$

con $a_i > 0$, è dato da:

$$x^* = \frac{\frac{1}{N}\sum_{i=1}^{N} a_i b_i}{\frac{1}{N}\sum_{i=1}^{N} a_i} \tag{11}$$

In presenza di funzioni di questo tipo è sufficiente definire le variabili $y_i(0) := a_i b_i$ e $z_i(0) := a_i$ ed in seguito aggiornarle con due algoritmi di *average consensus*:

$$\begin{array}{l}\mathbf{y}(k+1) = P\mathbf{y}(k) \\ \mathbf{z}(k+1) = P\mathbf{z}(k)\end{array} \qquad \mathbf{x}(k) = \frac{\mathbf{y}(k)}{\mathbf{z}(k)} \tag{12}$$

In questo modo si ha $\lim_{k\to\infty} \mathbf{x}(k) = x^* \mathbb{1}$, con P matrice di *consensus* doppiamente stocastica. Al tempo k abbiamo quindi per ogni agente una stima del minimo globale x^* data da :

$$x_i(k) = \frac{y_i(k)}{z_i(k)} \tag{13}$$

Nell'adottare questa idea per funzioni costo non quadratiche, si tiene conto dell'uguaglianza:

$$a_i b_i = f_i''(x)x - f_i(x) =: g_i(x) \tag{14}$$

$$a_i = f_i''(x) =: h_i(x) \tag{15}$$

ottenuta approssimando localmente la funzione con una parabola. Tuttavia non si può semplicemente impiegare l'algoritmo a partire da condizioni iniziali $y_i(0) = g_i(x_i(0))$ e $z_i(0) = h_i(x_i(0))$, dal momento che gli stati $x_i(k)$ variano in funzione del tempo k. Perciò, prima di ogni ciclo di *consensus*, il valore di $\mathbf{y}(k)$ e $\mathbf{z}(k)$ viene corretto in funzione del nuovo valore di $\mathbf{x}(k)$ come si vede nelle righe 7 e 8 dell'algoritmo originale in seguito riportato.
Inoltre, dal momento che inizialmente le stime del valore che minimizza la funzione di costo globale saranno molto distanti dal valore vero, ad ogni ciclo il valore nuovo di $\mathbf{x}(k)$ sarà una combinazione convessa del valore vecchio e di quello appena calcolato (riga 13 dell'Alg. 1), combinazione regolata dal parametro ε.

Algorithm 1 L'algoritmo di partenza (Zanella, Varagnolo, Cenedese, Pillonetto, Schenato; [1])

(variabili)
1: $\mathbf{x}(k), \mathbf{y}(k,m), \mathbf{z}(k,m) \in \mathbb{R}^N, m = 0,\dots,M; k = 0,1,\dots$

(parametri)
2: $P \in \mathbb{R}^{N\times N}$ matrice di consenso positiva e doppiamente stocastica
3: $\varepsilon \in (0,1)$

(initializzazione)
4: $\mathbf{x}(0) = \mathbf{0}, \mathbf{g}(\mathbf{x}(-1)) = \mathbf{h}(\mathbf{x}(-1)) = \mathbf{0}$
5: $\mathbf{y}(0,M) = \mathbf{z}(0,M) = \mathbf{0}$

(algoritmo)
6: **for** $k = 1,2,\dots$ **do**
7: $\quad \mathbf{y}(k,0) = \mathbf{y}(k-1,M) + \mathbf{g}(\mathbf{x}(k-1)) - \mathbf{g}(\mathbf{x}(k-2))$
8: $\quad \mathbf{z}(k,0) = \mathbf{z}(k-1,M) + \mathbf{h}(\mathbf{x}(k-1)) - \mathbf{h}(\mathbf{x}(k-2))$
9: $\quad$ **for** $m = 1,2,\dots,M$ **do**
10: $\quad\quad \mathbf{y}(k,m) = P\mathbf{y}(k,m-1)$
11: $\quad\quad \mathbf{z}(k,m) = P\mathbf{z}(k,m-1)$
12: $\quad$ **end for**
13: $\quad \mathbf{x}(k) = (1-\varepsilon)\mathbf{x}(k-1) + \varepsilon\frac{\mathbf{y}(k,M)}{\mathbf{z}(k,M)}$
14: **end for**

C. Ruolo del fattore di scala temporale ε

L'algoritmo illustrato nella sezione precedente può essere scritto in forma di sistema dinamico a tempo discreto:

$$\begin{cases} \mathbf{v}(k+1) = \mathbf{g}(\mathbf{x}(k)) \\ \mathbf{w}(k+1) = \mathbf{h}(\mathbf{x}(k)) \\ \mathbf{y}(k+1) = P[\mathbf{y}(k) + \mathbf{g}(\mathbf{x}(k)) - \mathbf{v}(k)] \\ \mathbf{z}(k+1) = P[\mathbf{z}(k) + \mathbf{h}(\mathbf{x}(k)) - \mathbf{w}(k)] \\ \mathbf{x}(k+1) = (1-\varepsilon)\mathbf{x}(k) + \varepsilon\frac{\mathbf{y}(k)}{\mathbf{z}(k)} \end{cases} \tag{16}$$

che può essere ricavato per discretizzazione con il metodo di Eulero in avanti, con passo di campionamento $T = \varepsilon$, a partire dal seguente sistema dinamico tempo continuo (si è posto $K = I - P^M$):

$$\begin{cases} \varepsilon\dot{\mathbf{v}}(t) = -\mathbf{v}(t) + \mathbf{g}(\mathbf{x}(t)) \\ \varepsilon\dot{\mathbf{w}}(t) = -\mathbf{w}(t) + \mathbf{h}(\mathbf{x}(t)) \\ \varepsilon\dot{\mathbf{y}}(t) = -K\mathbf{y}(t) + (I-K)[\mathbf{g}(\mathbf{x}(t)) - \mathbf{v}(t)] \\ \varepsilon\dot{\mathbf{z}}(t) = -K\mathbf{z}(t) + (I-K)[\mathbf{h}(\mathbf{x}(t)) - \mathbf{w}(t)] \\ \dot{\mathbf{x}}(t) = \mathbf{x}(t) + \frac{\mathbf{y}(t)}{\mathbf{z}(t)} \end{cases} \tag{17}$$

Per valori di ε sufficientemente piccoli quindi i due sistemi si comportano in maniera analoga. Il sistema scritto in questo modo è a tutti gli effetti un sistema dinamico a due scale temporali regolate da parametro ε, con una dinamica veloce espressa nelle prime quattro equazioni differenziali, ed una dinamica lenta che è costituita dall'ultima equazione. Sistemi di questo tipo vengono comunemente chiamati *standard singular perturbation models*, e se ne trova documentazione in [13]. Per ora ci limitiamo a osservare che per quanto riguarda le prime equazioni gli stati $\mathbf{y}(t)$ e $\mathbf{x}(t)$ si possono approssimare, per valori di t sufficientemente grandi, con

$$\mathbf{y}(t) \approx (\frac{1}{N}\mathbb{1}^T\mathbf{g}(\mathbf{x}(t)))\mathbb{1}$$

$$\mathbf{z}(t) \approx (\frac{1}{N}\mathbb{1}^T\mathbf{h}(\mathbf{x}(t)))\mathbb{1}$$

Inserendo queste equazioni nella dinamica lenta del sistema e tenendo presente la definizione di **g(x)** e **h(x)** in (5), si arriva all'approssimazione:

$$\mathbf{x}(t) \approx \overline{x}(t)\mathbb{1}$$

dove la quantità $\overline{x}(t)$ segue in maniera approssimata l'equazione differenziale:

$$\dot{\overline{x}}(t) = -\frac{\overline{f}'(\overline{x}(t))}{\overline{f}''(\overline{x}(t))} \tag{18}$$

si tratta quindi di un Alg. *Newton-Raphson* a tempo continuo che, nelle ipotesi nelle quali ci si è posti, converge al valore di minimo globale x^* di $\overline{f}(x)$.

La giustificazione formale delle osservazioni precedentemente sviluppate è data dalla seguente proposizione, presente nel *paper* originario [1]:

Proposizione 1: Si abbiano funzioni costo $f_i \in C^2\ \forall i$, siano cioè continue fino alla derivata seconda, con derivate seconde f_i'' strettamente positive, limitate e definite $\forall x \in \mathbb{R}$. Sia inoltre il valore di minimo globale $x^* \neq \pm\infty$.
Consideriamo l'Alg. (1), equivalente al sistema di equazioni (16) con condizioni iniziali $\mathbf{v}(0) = \mathbf{w}(0) = \mathbf{y}(0) = \mathbf{z}(0) = \mathbf{0}$.
Se le ipotesi precedenti sono soddisfatte, allora esiste $\overline{\varepsilon} \in (0, 1)$ tale che, se $\varepsilon < \overline{\varepsilon}$, l'Alg. (1), asintoticamente fa convergere ogni agente al valore di minimo globale x^*, quindi si ha $\lim_{t\to+\infty} \mathbf{x}(k) = x^*\mathbb{1}$.

Questo algoritmo è quindi valido per le specifiche condizioni iniziali descritte dalla proposizione precedente, ma è importante saggiare la robustezza di esso con condizioni iniziali diverse: la proposizione seguente stabilisce che il valore iniziale $x_i(0)$ può essere essere preso arbitrariamente, mentre le condizioni iniziali $v_i(0)$, $w_i(0)$, $y_i(0)$, $z_i(0)$ se sufficientemente elevate possono influire sulla stabilità e sulla convergenza del sistema.

Proposizione 2: Si consideri il sistema in (17) con condizioni iniziali arbitrarie $\mathbf{v}(0)$, $\mathbf{w}(0)$, $\mathbf{y}(0)$, $\mathbf{z}(0)$, $\mathbf{x}(0)$ e definiamo gli scalari:

$$\alpha(0) := \frac{1}{N}\mathbb{1}^T(\mathbf{y}(0) - \mathbf{v}(0)) \tag{19}$$

$$\beta(0) := \frac{1}{N}\mathbb{1}^T(\mathbf{z}(0) - \mathbf{w}(0)) \tag{20}$$

Se valgono le ipotesi della proposizione 1 allora esistono $\overline{\varepsilon}$, $\overline{\alpha}$, $\overline{\beta} \in \mathbb{R}_+$ tali che, se $\varepsilon < \overline{\varepsilon}$, $|\alpha(0)| < \overline{\alpha}$, $|\beta(0)| < \overline{\beta}$ si ha $\lim_{t\to+\infty} \mathbf{x}(t) = \xi(\alpha(0), \beta(0))\mathbb{1}$, dove $\xi(\alpha(0), \beta(0))$ è una funzione scalare continua nei suoi argomenti tale per cui $\xi(0, 0) = x^*$.

D. Ulteriori considerazioni sull'algoritmo presentato

1) Algoritmo di Newton-Raphson: Richiamiamo ora brevemente l'algoritmo di Newton-Raphson classico, che è alla base dell'algoritmo distribuito che stiamo studiando. Siano a, b gli estremi dell'intervallo in cui cade la radice dell'equazione ed assumiamo f con derivata prima e seconda a segno costante nell'intervallo in questione. Si può notare che $|f'(x)|$ è crescente quando il segno delle due derivate è lo stesso, decrescente quando è discorde. In ogni caso $|f'(x)|$ raggiunge il valore massimo nell'estremo nel quale $f(x)$ ha lo stesso segno di $|f''(x)|$: indichiamo questo valore con x_0, quello di partenza dell'algoritmo (chiamato in letteratura estremo di Fourier).
Sia ora x_1 l'ascissa del punto in cui la tangente al grafico in $(x_0, f(x_0))$ interseca l'asse x, si può dimostrare che $x_1 \in (x^*, x_0)$ e si ha, per la positività di $f(x)$ e $f'(x)$:

$$\frac{f(x_0)}{f'(x_0)} > 0$$

perciò:

$$x_1 - x_0 = -\frac{f(x_0)}{f'(x_0)} < 0$$

quindi $x_1 < x_0$: il punto x_1 approssima meglio x^* rispetto ad x_0. È possibile iterare il procedimento partendo dal nuovo punto calcolato, in generale:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \qquad k = 1, 2, ...$$

e si può dimostrare la convergenza per $k \to +\infty$ della successione x_k alla radice x^* (allo scopo si veda un qualsiasi testo di calcolo numerico).

2) Gossip consensus: In un algoritmo di consenso, la comunicazione tra i nodi può anche avvenire secondo uno schema casuale, ovvero ad ogni istante temporale vengono scelti casualmente due nodi che comunicano tra loro ed attuano la media dei loro stati privati. In questo caso si parla di *Gossip consensus*.
Gli algoritmi *Gossip* possiedono interessanti proprietà perché si è dimostrato che in uno schema di comunicazione riducono il numero delle trasmissioni necessarie rispetto agli algoritmi deterministici ed evitano errori e collisioni nello scambio dei dati; per un maggior dettaglio si veda [3] e [8].
Sia $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ un grafo con N nodi, completo e non orientato, con lati $\mathcal{E} \subset \{(i, j) \in \mathcal{V}\}$. Ad ogni istante temporale un lato (i, j) viene scelto casualmente in $\mathcal{E}$ con probabilità $W^{(i,j)} > 0$ tale che $\sum_{(i,j)\in\mathcal{E}} W^{(i,j)} = 1$.
I due nodi connessi attuano la media dei loro stati locali:

$$\begin{cases} x_i(k+1) = \frac{x_i(k)+x_j(k)}{2} \\ x_j(k+1) = \frac{x_j(k)+x_i(k)}{2} \end{cases} \tag{21}$$

mentre gli altri nodi restano inalterati:

$$x_h(k+1) = x_h(k) \qquad \forall h \neq i, j \tag{22}$$

Sia ora $e_i = [0, \ldots, 0, 1, 0, \ldots, 0]$ il vettore $N \times 1$ con l'*i*_esima componente uguale ad 1, sia $E_{ij} = (e_i - e_j)(e_i - e_j)^T$ e definiamo la matrice di *consensus*:

$$P(k) = I - \frac{E_{ij}}{2}$$

Le equazioni (21) e (22) si possono allora riscrivere in modo compatto:

$$\mathbf{x}(k+1) = P(k)\mathbf{x}(k)$$

dove $\mathbf{x}(k) = [x_1(k), ..., x_N(k)]$ rappresenta lo stato del sistema. Si vede che $P(k)$ è doppiamente stocastica, quindi, con le ipotesi assunte sul grafo, il sistema converge asintoticamente con probabilità uno al valore dell'*average consensus*:

$$\lim_{t\to+\infty} \mathbf{x}(k) = x^*\mathbb{1} = \frac{\mathbb{1}^T\mathbb{1}}{N}\mathbf{x}(0)$$

Si può dimostrare inoltre che questo è l'unico algoritmo randomizzato che preserva la media degli stati iniziali.

3) Distributed Newton-Raphson Gossip consensus: Vediamo ora in che modo il *Gossip consensus* appena presentato si può applicare all'algoritmo illustrato in II. La struttura è simile a quella presentata in Alg. 1, con la differenza che all'interno del ciclo **for** (alla riga 9) la matrice P non è costante, ma cambia ad ogni iterazione. Lo scopo è quello di simulare uno schema di comunicazione che richieda meno

sincronismo tra i nodi della rete, risultando quindi realisticamente più simile ai casi concreti di applicazione.

Per la matrice P si è scelto di adottare lo schema del *gossip simmetrico* in modo da garantire automaticamente la convergenza alla media del *consensus*. Ad ogni iterazione un certo nodo può comunicare esclusivamente con un solo altro nodo e le coppie di nodi che interagiscono vengono scelte in modo casuale. Data una tipologia ad anello, ogni nodo comunicherà con uno dei suoi vicini secondo una probabilità decrescente con la distanza che li separa (intesa come numero di archi). Questa è una soluzione realistica in quanto la casualità rappresenta la possibilità che alcuni agenti smettano di comunicare tra loro per un certo periodo, mentre la distribuzione di probabilità proposta rispecchia il fatto che è molto verosimile che i nodi limitrofi interagiscano spesso tra loro, ma non esclude la possibilità che si instauri una comunicazione anche tra interlocutori più distanti. Si è scelto di permettere una comunicazione tra i nodi che distanto al massimo 4 archi tra loro secondo la seguente distribuzione:

$$\mathbf{p} = [0.2, 0.15, 0.1, 0.05] \tag{23}$$

dove ogni elemento p_i del vettore **p** indica la probabilità di comunicare con un nodo che dista i archi su una tipologia ad anello. Infine si è scelto un *consensus weight* pari a $w = 0.25$. In Alg. 2 è illustrato lo pseudocodice della procedura con cui viene creata la matrice 'P ad ogni iterazione[1].

In Fig. 1 è rappresentata l'evoluzione temporale degli stati

Algorithm 2 Algoritmo per la determinazione della matrice di *consensus* P

(variabili)
$p = [0.2, 0.15, 0.1, 0.05]$ ▷ distribuzione di probabilità
$N = 10$ ▷ numero di agenti
$w = 0.25$ ▷ *consensus weight*
$C = \{$insieme degli N nodi$\}$
for $r = 1, \ldots, N$ **do** ▷ per ogni riga di P
  if r è in C **then**
    scegli un nodo c in C, diverso da r, secondo la distribuzione p
    $P(r, r) = 1 - w$
    $P(r, c) = w$
    $P(c, c) = 1 - w$
    $P(c, r) = w$
    togli c da C
  end if
end for

dell'esempio proposto in [1], ovvero con funzioni costo somme di esponenziali del tipo:

$$f_i(x) = c_i e^{a_i x} + d_i e^{-b_i x}, \quad i = 1, ..., N \tag{24}$$

$$a_i, b_i \sim \mathcal{U}[0, 0.2], \quad c_i, d_i \sim \mathcal{U}[0, 1]. \tag{25}$$

[1]Esso va considerato racchiuso all'interno del ciclo **for** presente in Alg. 1.

Il risultato è paragonabile a quello ottenuto con la versione sincrona dell'algoritmo. In conclusione, è stato possibile ricavare facilmente un'implementazione asincrona con poche e semplici modifiche e ottenere comunque le stesse prestazioni.

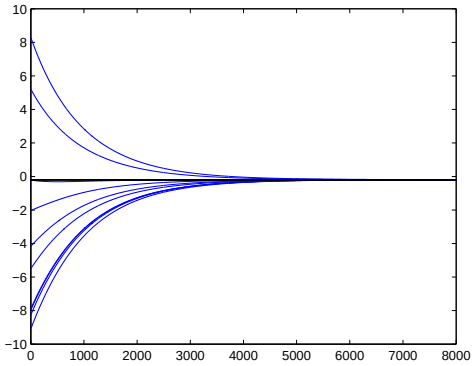

Fig. 1. Evoluzioni temporali degli stati $x_i(k), i = 1, ..., N$ per $N = 10$ e coefficienti come in 25 ottenute con ε costante pari a 0.001, di poco inferiore a ε^*.

III. Applicazioni

Gli algoritmi di *consensus* sono recentemente oggetto di studio intenso perché il controllo distribuito sta avendo un sempre maggior impatto in ambito civile, industriale e militare. Si tende, infatti, a sostituire costosi controlli centralizzati, sensori ed attuatori con dispositivi che possano autonomamente compensare potenziali errori e guasti grazie a strategie di comunicazione e cooperazione tra di essi.
Potenziali applicazioni riguardano il monitoraggio di foreste per la prevenzione di incendi, di pozzi petroliferi e *pipelines*, di sorveglianza di zone sensibili quali, tra gli altri, obiettivi militari ed il controllo del traffico aereo e stradale. Gli algoritmi di *consensus* possono trovare inoltre notevoli impieghi anche nel contesto di reti di sensori per l'osservazione, ad esempio, di caratteristiche ambientali quali la temperatura, il grado di inquinamento, l'umidità etc.
Nel seguito verranno illustrati alcuni esempi di possibili applicazioni dell'algoritmo oggetto della relazione.

A. Stima distribuita di parametri

Consideriamo una rete composta da *n* sensori omogeneamente distribuiti in un metro quadro, ciascuno dei quali rileva *m* misure. Supponiamo che l'obiettivo sia quello di ricavare il valore medio di tutte queste misure che rappresenta un parametro di nostro interesse, come ad esempio la temperatura media di un ambiente, l'inquinamento medio di una zona etc. Possiamo considerare tre diversi approcci:

1) i sensori trasmettono tutti i dati ad un processore centrale che ne calcola la media: avremo *O(mn)* bit trasmessi lungo una distanza media di *O(1)* metri;
2) i sensori calcolano autonomamente la media locale e la trasmettono al processore centrale che elabora quindi il

valore medio globale: stavolta i bit trasmessi saranno *O(n)* su *O(1)* metri;

3) definiamo un percorso che passi attraverso tutti i nodi, visitandoli solo una volta e saltando dal nodo corrente al più vicino ad esso. La media globale può essere calcolata durante il percorso di visita: il nodo corrente aggiunge la sua media locale al valore totale accumulato fino a quel punto. Alla fine avremo quindi un totale proporzionale alla media globale. I bit trasmessi, tra i nodi, saranno con evidenza *O(n)* ma il percorso compiuto è mediamente solo di $O(n^{-1/2})$ metri.

L'ultimo approccio è quindi molto più efficiente dei due precedenti. La media può essere vista come il valore che minimizza funzioni costo quadratiche, ossia come un problema di ottimizzazione convessa del tipo:

$$x^* = \underset{x}{\operatorname{argmin}} \overline{f}(x) = \underset{x}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} f_i(x)$$

dove x^* è il parametro da stimare, $\overline{f}(x)$ è la funzione di costo globale da minimizzare data dalla somma di *n* funzioni costo $f_i(x)$ che dipendono dalle misure rilevate dall'*i*-esimo sensore. Se $x_{i,j}$ è la *j*-esima misura fornita dall'*i*-esimo sensore, continuando l'esempio avremo:

$$\overline{f}(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x) \qquad f_i(x) = \frac{1}{m} \sum_{j=1}^{m} (x_{i,j} - x)^2$$

L'algoritmo agisce passando una stima di x^* nodo per nodo, il quale aggiorna il parametro riducendo il proprio costo locale e passando la nuova stima al nodo successivo che ripeterà l'operazione. Asintoticamente si raggiungerà così il valore di minimo x^*. Si può notare inoltre che, oltre alla maggiore velocità computazionale, l'algoritmo di *consensus* permette anche un minor dispendio in termini energetici rispetto ad un controllo centralizzato, specialmente nel caso in cui si abbiano sensori molto distanti tra di loro.

B. Problemi di regressione lineare e robusta

Un'analisi di regressione è un problema nel quale si cerca di trovare una funzione che si avvicini il più possibile ad un insieme di dati rilevati, tipicamente punti nel piano.
Il nostro algoritmo può essere efficacemente impiegato allo scopo, infatti la funzione di regressione di interesse può essere parametrizzata nella forma:

$$f_\theta(x) = \sum_{i=1}^{m} \theta_i h_i(x)$$

dove $h_i(x)$ sono funzioni note e $\theta = [\theta_1 \theta_2 \dots \theta_m]^T$ sono parametri incogniti determinabili in base ai dati rilevati $\{(x_i, y_i)\}_{i=1}^{N}$. I parametri incogniti si possono calcolare attraverso la formula:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{N} (y_i - f_\theta(x_i))^2$$

Le formule appena introdotte utilizzano la classica funzione di costo quadratica $\|y - f(x)\|^2$ per risolvere il problema di ottimizzazione convessa. Un problema di minimizzazione ai minimi quadrati si risolve in modo immediato con semplici routine numeriche, come descritto nel libro di G. Picci [16], l'unica ipotesi importante è che la funzione obiettivo sia quadratica (e che la forma quadratica associata sia semidefinita positiva).

Nelle reti di sensori ci sono molte ragioni che portano alla scelta di schemi alternativi e quindi di funzioni costo differenti dalla classica funzione costo quadratica. Un esempio esplicativo può essere il seguente.
Supponiamo di dover monitorare l'inquinamento di un'area urbana attraverso una rete di *n* sensori, ognuno dei quali rileva *m* misure durante il giorno e, alla fine della giornata, si stabilisce che il valore dell'inquinamento complessivo è la media delle misure raccolte.
Se la varianza d'errore di ogni misura vale σ^2, assumendo per semplicità distribuzioni dei campioni indipendenti identicamente distribuiti (i.i.d), il valore finale di inquinamento avrà varianza $\frac{\sigma^2}{mn}$. Se però avessimo, ad esempio, il 10% dei sensori danneggiati o calibrati male con varianze d'errore di $100\sigma^2$, la varianza della misura finale crescerebbe ben di dieci volte. Ecco il motivo che ci spinge ad utilizzare funzioni costo diverse dalla quadratica, più "robuste" nel tener conto di possibili errori. Entriamo quindi nel campo della regressione robusta.

Le funzioni di costo quadratiche sono infatti particolarmente sensibili alla presenza di *outliers*, ossia di valori anomali ed inconsueti se paragonati al resto delle osservazioni. I metodi di regressione robusta hanno lo scopo di rendere quanto meglio ininfluenti gli eventuali *outliers* attraverso funzioni costo appositamente studiate: le più note sono quella di *Huber*, L^1 (o *Laplace*), ε-*insensitive* (chiamata anche *deadzone-linear*) e *log-barrier* (si veda allo scopo Fig.4 ed i testi [6] e [11]).
In Fig. III-B si può apprezzare la differenza di risultati che si ottengono in un problema di regressione se viene utilizzata una funzione di costo quadratica (linea tratteggiata rossa) e la funzione di costo di L^1 (linea continua nera): è evidente l'influenza che hanno gli *outliers* (cerchi rossi) nel primo caso, mentre nel secondo caso la retta di regressione appare meno condizionata dagli *outliers*; allo scopo la si confronti con la retta "pulita" a puntini di colore blu.

Il concetto di regressione robusta verrà ripreso nel seguito della relazione con l'analisi dettagliata dell'algoritmo nel caso di impiego di funzioni costo robuste del tipo L^1.

C. Allocazione di risorse

Un'altra possibile applicazione del nostro algoritmo può riguardare il campo dell'allocazione ottimale di risorse per problemi computazionali. Si pensi ad esempio a reti *internet* con decine di unità di calcolo connesse tra loro: è evidente l'utilità, anche in termini economici, di un algoritmo che ottimizzi le prestazioni computazionali dell'intera rete.

Siano stavolta i processori di un sistema i nodi del grafo mentre i lati di esso siano le connessioni fisiche presenti tra

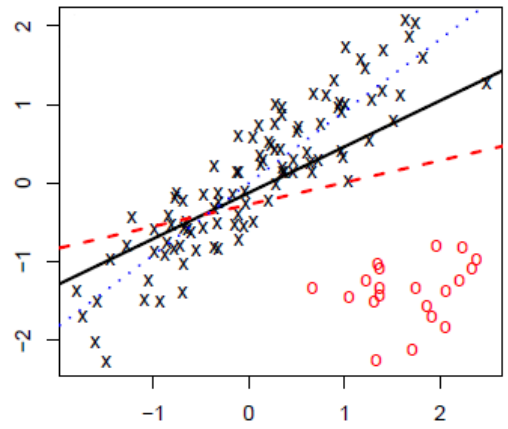

Fig. 2. Confronto delle rette di regressione di un set di dati buoni (crocette nere) e *outliers* (cerchi rossi) ottenute tramite l'utilizzo della funzione costo quadratica (linea rossa tratteggiata) e della funzione costo L^1 (linea nera). La retta blu a puntini invece è il risultato ottenuto con la funzione costo quadratica senza la presenza degli *outliers*.

i nodi. La misura x_i di ogni nodo potrebbe essere in questo caso il numero di operazioni che l'*i*_esimo processore deve svolgere per una determinata operazione di calcolo ai fini di una elaborazione globale che comprenda l'intera rete. Per migliorare complessivamente la velocità di calcolo si potrebbe fare in modo che i processori possano distribuirsi omogeneamente le operazioni la svolgere sfruttando le connessioni presenti in modo da bilanciare il lavoro totale. L'obiettivo è quindi quello che tutti i processori abbiano alla fine lo stesso numero di operazioni da svolgere, ossia la quantità media x^*. Un approccio possibile può essere quindi l'utilizzo di un algoritmo di *consensus* distribuito che tenga conto della tipologia della rete attraverso un grafo e pesi il numero delle operazioni di calcolo in carico a ciascun processore con delle apposite funzioni costo. Ogni nodo aggiornerà il valore medio x^* tramite un algoritmo di *consensus*, successivamente avverrà lo spostamento effettivo delle operazioni di calcolo tra i processori.
Matematicamente si tratta di determinare x^* che minimizza il carico totale di operazioni dell'intera rete; se definiamo con $f_i(x)$ le funzioni costo di ogni nodo, si perviene al problema:

$$x^* = \underset{x}{\operatorname{argmin}} \sum_{i=1}^{N} f_i(x)$$

Con evidenza questo problema può essere risolto con l'algoritmo in esame.

D. Localizzazione di una fonte acustica

La localizzazione di una fonte acustica è un particolare problema che si riscontra anche in ambito militare.
Supponiamo che la fonte acustica sia in una posizione ignota θ^* da stimare attraverso delle misure rilevate da una serie di sensori distribuiti uniformemente nello spazio. Se r_i è la locazione, nota, dell'*i*_esimo sensore ($i = 1, ..., N$) che rileva m misure, possiamo descrivere il segnale ricevuto utilizzando il modello isotropo di propagazione dell'energia:

$$x_{i,j} = \frac{A}{||\theta - r_i||^\beta} + w_{i,j} \qquad j = 1, ..., m$$

dove $A > 0$ è una costante, $||\theta - r_i||^\beta > 1$, l'esponente $\beta > 1$ descrive l'attenuazione acustica causata dal mezzo di propagazione e $w_{i,j}$ sono supposti rumori di distribuzione gaussiana a media nulla e varianza σ^2.
La locazione della fonte rumorosa si trova risolvendo:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \frac{1}{mN} \sum_{i=1}^{N} \sum_{j=1}^{m} (x_{i,j} - \frac{A}{||\theta - r_i||^\beta})^2$$

Si tratta evidentemente di un problema di minimizzazione (non lineare) ai minimi quadrati, sul quale possiamo applicare il nostro algoritmo ponendo come funzioni costo:

$$f_i(\theta) = \frac{1}{m} \sum_{j=1}^{m} (x_{i,j} - \frac{A}{||\theta - r_i||^\beta})^2$$

E. Nodi in una rete

In alcuni casi può essere utile conoscere quanti nodi sono presenti in una rete (si pensi ad esempio al caso di *PC* collegati alla rete *Internet* oppure sensori che trasmettono informazioni etc.). Questa operazione può essere fatta semplicemente con un algoritmo di *average consensus*, inizializzando un nodo ad 1, $x_1(0) = 1$, e tutti gli altri a zero, $x_i(0) = 0$, $i = 2, 3, \ldots, N$. Siccome l'*average consensus* converge alla media delle condizioni iniziali, ponendo la variabile:

$$\hat{N}_i(k) = \frac{1}{x_i(k)}$$

che stima N in ogni nodo i della rete, otteniamo:

$$\hat{N}_i(k) = \frac{1}{x_i(k)} \xrightarrow{k \to +\infty} \frac{1}{\frac{1}{N} \sum_{j=1}^{N} x_j(0)} = N$$

essendo ovviamente $\sum_{j=1}^{N} x_j(0) = 1$ come prima specificato.

F. Calibrazione di sensori

Molte tipologie di sensori economici spesso possono essere affetti da errori di offset dovuti ad imprecisioni di fabbricazione. Se si utilizzano dispositivi che misurano la potenza del segnale (indice *RSII*) per stimare la distanza tra due nodi wireless, la misura di potenza del *j*-esimo dispositivo rilevata dall'*i*-esimo dispositivo può essere modellizzata come:

$$y_{ij} = f(\xi_i, \xi_j) + o_i$$

dove ξ_i e ξ_j sono la posizione dei due dispositivi, o_i è l'offset del dispositivo ricevente e $f(\xi_i, \xi_j)$ è di solito funzione della distanza $||\xi_i - \xi_j||$ tra gli apparecchi.

L'obiettivo è quello di stimare l'offset o_i di ogni nodo con lo scopo di rimuoverlo dalle misurazioni. Possiamo impostare il problema come segue: vogliamo stimare l'offset $\hat{o}_i$ in modo tale che per ogni nodo valga la relazione $o_i - \hat{o}_i = \overline{o}$, ossia che tutti i nodi abbiano lo stesso offset $\overline{o}$.

Ci troviamo di fronte ad un problema di *average consensus* per la variabile:

$$x_i(t) = o_i - \hat{o}_i$$

e si deve trovare il valore $\overline{o}$:

$$\begin{aligned} \underset{\overline{o}}{\operatorname{argmin}} \sum_{i=1}^{N} \hat{o}_i^2 &= \underset{\overline{o}}{\operatorname{argmin}} \sum_{i=1}^{N} (o_i - \overline{o})^2 \\ &= \frac{1}{N} \sum_{i=1}^{N} o_i = \frac{1}{N} \sum_{i=1}^{N} x_i(0) \end{aligned} \tag{26}$$

dove si è posto $\hat{o}_i(0) = 0$; sostituendo nell'ultimo membro l'equazione $x_i(k) = o_i - \hat{o}_i(k)$, abbiamo:

$$o_i - \hat{o}_i(k+1) = o_i - \hat{o}_i(k) + \sum_{j=1}^{N} p_{i,j}(k)[(o_j - \hat{o}_j(k)) - (o_i - \hat{o}_i(k))]$$

$$\hat{o}_i(k+1) = \hat{o}_i(k) - \sum_{j=1}^{N} p_{i,j}(k)[(f_{ij} + o_j - \hat{o}_j(k)) - (f_{ji} + o_i - \hat{o}_i(k))]$$

$$= \hat{o}_i(k) + \sum_{j=1}^{N} p_{i,j}(k)[\hat{o}_j(k) - \hat{o}_i(k) + y_{ij} - y_{ji}]$$

e si è assunta l'ipotesi $f(\xi_i, \xi_j) = f(\xi_j, \xi_i) = f_{ij} = f_{ji}$.

Applicando l'algoritmo di *average consensus* si ottiene infine:

$$\hat{o}_i(k) \xrightarrow{k \to +\infty} o_i - \frac{1}{N} \sum_{j=1}^{N} o_j$$

Ipotizzando realisticamente che l'offset sia distribuito normalmente, $o_i \sim \mathcal{N}(0, \sigma^2)$, per N (il numero dei dispositivi) sufficientemente grande l'ultimo addendo può essere allora trascurato, quindi la stima dell'offset ottenuta può ritenersi molto vicina al valore vero.

IV. Funzioni costo e outliers

L'algoritmo proposto in [1] può essere adottato nel caso si abbia a che fare con funzioni costo sufficientemente regolari. Storicamente le funzioni costo più comuni sono quelle quadratiche, in quanto la risoluzione di un problema ai minimi quadrati è molto facile da implementare per via numerica, e in genere si riconduce ad un problema lineare [16].

Nel più generico caso multivariabile, le funzioni costo quadratiche possono essere scritte nella forma:

$$f_i(x) = \frac{1}{2}(x - b_i)^T A_i (x - b_i) \tag{27}$$

con $x \in \mathbb{R}^n$ e $A \in \mathbb{R}^{n \times n}$ definita positiva. Il minimo di $f_i(x)$ si ha per $x = b_i$ e vale 0. Nel caso in cui si voglia trovare il valore x^* che minimizza una somma di S funzioni costo del tipo (27), definita la funzione costo globale:

$$\overline{f}(x) = \sum_{i=1}^{S} f_i(x) \tag{28}$$

il valore minimo di $\overline{f}(x)$ si ha per:

$$x^* = \left(\frac{1}{S} \sum_{i=1}^{S} A_i \right)^{-1} \left(\frac{1}{S} \sum_{i=1}^{S} A_i b_i \right) \tag{29}$$

La formula (29) è alla base dell'estensione al caso multivariabile dell'algoritmo illustrato nella Sez. II, per $n = 1$ infatti ci si riduce alla (11).

Si può osservare come la (29) sia estremamente conveniente, in termini di impiego di risorse di calcolo, qualora si voglia trovare esplicitamente il minimo globale in una sola riga di calcolo. Infatti è necessario operare unicamente un inversione di matrice oltre ad operazioni elementari quali moltiplicazioni e addizioni.

Diversamente, con altre strutture delle funzioni costo, ci si troverebbe di fronte a calcoli molto più complicati per il calcolo del minimo. Oltre ai vantaggi che introduce dal punto di vista del calcolo, questo tipo di funzione costo presenta un problema di scarsa robustezza nei confronti dei cosiddetti outliers, ovvero i valori anomali all'interno dei dati osservati [11].

Un'esempio grafico di outlier si trova in Fig. 3.

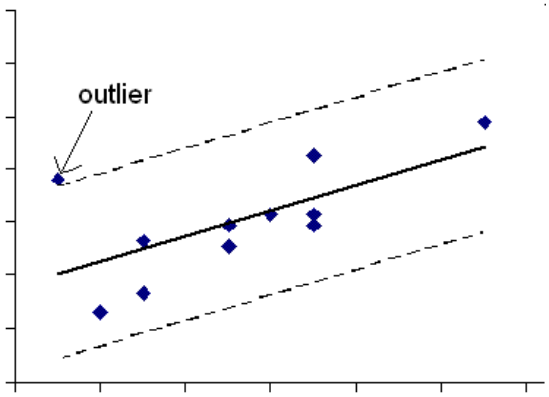

Fig. 3. Illustrazione che rappresenta la regressione lineare di un insieme di dati in presenza di un possibile outlier.

Supponiamo che si voglia effettuare una regressione lineare a partire da un insieme di dati rumorosi, come quelli in Fig. 3. I dati al di fuori di un certo intervallo di confidenza portano meno informazione circa la vera natura della distribuzione che ha generato i dati, perciò si vorrebbe non considerarli nella regressione lineare: infatti, in particolare modo nel caso di funzioni costo quadratiche, dati molto distanti dal modello "vero" portano a stime sbagliate, in quanto la distanza dalla retta di regressione viene elevata al quadrato.

Lo studio di queste problematiche può essere trovato in diversi testi, quali ad esempio [11], da cui è tratta la Fig. 4. In questa sede ci limiteremo unicamente a citarle, e adotteremo le soluzioni proposte in [11] per riuscire ad applicare l'algoritmo di consenso per l'ottimizzazione di funzioni convesse in presenza di outliers.

Al fine di evitare situazioni di questo tipo, vengono usate spesso funzioni che crescono in modo lineare man mano che

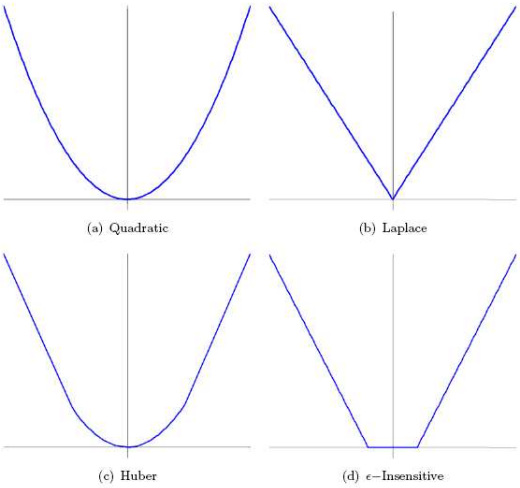

Fig. 4. Diversi tipi di funzioni costo presenti in letteratura.

ci si allontana dal minimo: fanno parte di questa categoria quelle rappresentate nei grafici (b) (c) e (d) in Fig. 4. In questo elaborato verrà affrontato un problema di ottimizzazione in presenza di outliers, attraverso l'utilizzo di funzioni riconducibili a quelle di tipo (c) e (d). Per poter effettuare i calcoli delle derivate prime e seconde necessari per far funzionare l'algoritmo illustrato nella Sez. II, si è scelto di considerare funzioni iperboliche, o piè precisamente dei rami di iperbole, che sono, tra le funzioni analitiche note, quelle che più si prestano all'approssimazione di (c) e (d) per valori lontani dallo zero.

Nella prossima sezione vengono presentate le funzioni costo iperboliche, ed il modo in cui queste vengono implementate all'interno dell'algoritmo illustrato in II.

A. Funzioni costo iperboliche

Per semplificità consideremo l'utilizzo di funzioni costo del tipo (d), e le approssimeremo con dei rami di iperbole. Nell'esempio di mapping, che illustreremo in seguito, le funzioni costo associate ai diversi agenti sono tutte uguali, in quanto si dà uguale importanza all'errore di approssimazione che si commette presso ciascun agente. Questo permette di semplificare notevolmente l'approssimazione delle funzioni costo. Si suppone che ciascuna funzione costo $f_i(x)$ abbia il valore minimo, corrispondente all'ordinata della parte "piatta", pari a 0, e che l'ascissa del centro del suo centro di simmetria sia anch'essa pari a zero. Di conseguenza ci si trova a che fare con delle strutture che hanno due gradi di libertà:

- la pendenza dei due rami $\pm m$, con $m \in]0, +\infty[$;
- la larghezza della parte centrale a costo nullo.

Ora, dal momento che le nostre funzioni $f_i(x)$ sono tutte uguali, si impone che gli asintoti dell'iperbole abbiano pendenza $\pm m$, in questo modo l'ipotesi $f_i(x) \in C^2,\ \forall i$ viene preservata, le funzioni costo sono ancora convesse, e di conseguenza si ha la garanzia della convergenza dell'algoritmo. Inoltre non siamo troppo interessati all'andamento delle funzioni per valori centrali, bensì ci basta sapere che il punto di minimo si trovi nello zero per tutte le funzioni $f_i(x)$[2]. Questa scelta può sembrare arbitraria ma non incide sulla qualità del lavoro che svolgeremo, la nostra attenzione infatti è sul contrasto degli outliers, e questo risultato è legato alla crescita lineare e non quadratica che si ha allontanandosi dall'origine.

Volendo ricavare la forma analitica delle funzioni costo, ricordiamo che l'iperbole I è il luogo geometrico dei punti del piano tali che la differenza in valore assoluto delle distanze da due punti fissi F_1 e F_2 detti fuochi è costante, ovvero, $\forall P \in I$, si ha $\left|\overline{PF_1} - \overline{PF_2}\right| = k$, con k costante. Dal momento che la funzione $f(x)$ del tipo (d) che stiamo approssimando è centrata nell'origine, cerchiamo di approssimarla con il ramo superiore appartenente ad un'iperbole di centro l'origine e fuochi sull'asse y, che com'è noto ha equazione canonica

$$\frac{x^2}{\alpha^2} - \frac{y^2}{\beta^2} = -1 \tag{30}$$

con α e β legate dalla relazione $\alpha^2 + \beta^2 = \gamma^2$, dove γ è il modulo dell'ordinata dei due fuochi ($F_1 = (0, \gamma)$, $F_2 = (0, -\gamma)$).

Si impone che gli asintoti dell'iperbole abbiano pendenza $\pm m$. In questo modo, considerando solo il ramo superiore dell'iperbole, la funzione costo diventa:

$$f_i(x) = \beta\sqrt{\frac{x^2}{\alpha^2} + 1} \tag{31}$$

Ricordando che le equazioni per gli asintoti sono $y = -\frac{\beta}{\alpha}$ e $y = \frac{\beta}{\alpha}$, e scegliendo in maniera arbitraria $\gamma > 0$, si perviene infine alle due formule per i parametri α e β:

$$\begin{aligned} \alpha &= \sqrt{\tfrac{\gamma}{1+m^2}} \\ \beta &= m \cdot \alpha \end{aligned} \tag{32}$$

Nel proseguio di questo lavoro, spesso capiterà che gli agenti del nostro sistema debbano calcolare i valori di una determinata funzione costo $f(x_i)$ o delle sue derivate prime e seconde, a partire dal dato x_i a disposizione di ciascun agente, che sarà in generale un errore di approssimazione. In questi casi, essendo il ramo di iperbole una funzione in C^∞ (e quindi in C^2), i valori saranno ricavati da ciascun agente secondo le espressioni:

$$g(x_i) = \beta\sqrt{\frac{x^2}{\alpha^2} + 1} \tag{33}$$

$$g'(x_i) = \frac{\beta^2}{\alpha^2}\frac{2x_i}{g(x_i)} \tag{34}$$

$$g''(x_i) = 2\frac{\beta^2}{\alpha^2}\frac{g(x_i) - x_i g'(x_i)}{g(x_i)^2} \tag{35}$$

[2]D'ora in poi si ometterà l'indice e si farà riferimento alle funzioni costo con la scrittura $f(x)$.

V. Esempio pratico dell'utilizzo di funzioni costo non quadratiche: Mapping

In questa sezione si porta l'esempio di un caso pratico in cui l'utilizzo di funzioni costo non quadratiche garantisce risultati migliori rispetto alla scelta di funzioni di tipo quadratico.

Si vuole ricostruire l'andamento di una funzione scalare (quindi si vuole creare una mappa) a partire da una serie di misure rumorose (ad esempio delle misure di temperatura, umidità, ecc. lungo una sola direzione, corrispondente alla variabile x). Chiameremo la mappa $m(x)$. L'espressione analitica[3]. Le differenti espressioni analitiche che generano le due protuberanze saranno significative al momento di valutare la qualità della ricostruzione, nelle sezioni V-A e V-B. Considereremo i valori della mappa $m(x)$ nei punti x_i, $i = 0, \ldots, N-1$, dal momento che la si valuta solo nei punti dove sono disposti i sensori, si può considerare quindi $m(x)$ come una funzione discreta $m(x_i)$.

Si vuole approssimare la mappa $m(x)$ con la somma di un certo numero di funzioni regolari. In particolare illustreremo il caso in cui le funzioni utilizzate per rappresentare la mappa sono delle "campane" del tipo:

$$g_i(x) = a_i e^{\frac{(x-w_i)^2}{R}}, \qquad R > 0 \tag{36}$$

centrate in M punti w_i, $(i = 0, .., M-1)$, disposti a intervalli regolari lungo tutta la lunghezza della mappa da ricostruire. Il fattore R verrà omesso nel proseguio per non appesantire la notazione: il suo ruolo è quello di "allargare" la campana, in modo da ottimizzare le risorse per la ricostruzione della mappa. Se il profilo che si vuole ricostruire presenta numerosi picchi e variazioni repentine, si imporrà che il parametro R sia piccolo, e allo stesso tempo si posizioneranno i centri delle campane più vicini l'un l'altro, il che necessariamente aumenta la quantità di informazione da elaborare e quindi la potenza di calcolo necessaria. Profili che variano in maniera più graduale, al contrario, permettono di "allargare" le campane, e diminuirne il numero, snellendo la complessità dell'algoritmo stesso. Un'ulteriore considerazione sulle funzioni utilizzate per l'operazione di mapping riguarda la scarsa influenza che ciascuna campana ha sui punti a sufficiente distanza dal proprio centro. Essendo realizzate con esponenziali, infatti, l'effetto della campana nella ricostruzione della mappa è pressoché nullo nei sensori più distanti[4]. In questo lavoro, tuttavia, non ci si è preoccupati di ottimizzare questo aspetto, che potrebbe essere spunto per un ulteriore approfondimento.

[3]l'espressione integrale è $m(x) = \frac{1}{20}(0.8\sin(\frac{x}{5}) + 3\frac{x-4}{1+(x-2)^2} + \frac{5}{1+x^4} + 5 + 3\cos(\frac{x}{10} - \frac{\pi}{4}) + \frac{1}{100(x-15)} + 50e^{-\frac{(x-15)^2}{50}} - \frac{1}{2}\frac{x}{1+(\frac{x-100}{10})^2})$ della mappa utilizzata nell'esempio non è particolarmente rilevante, se non per sottolineare che le due grosse "protuberanze" presenti sono dovute ai termini $50e^{-\frac{(x-15)^2}{50}}$ e $-\frac{1}{2}\frac{x}{1+(\frac{x-100}{10})^2}$. Non c'è giustificazione particolare della sua complicatezza, se non quella di garantire una sufficiente irregolarità.

[4]Ovviamente questo dipenderebbe anche dal parametro R, tuttavia d'ora in poi lo considereremo normalizzato pari a $R = 1$, ovvero si suppone che la scala del nostro problema sia tale che una campana ottima abbia coefficiente $R = 1$.

Nel caso di ottimizzazione centralizzata, alla k-esima iterazione, la nostra ricostruzione della mappa $m(x)$ sarà pari a

$$G(x, \mathbf{a}(k)) = \sum_{i=1}^{M} g_i(x, a_i(k)) \tag{37}$$

Il problema sarà quindi un problema di ottimizzazione, ovvero si cerca il vettore $\mathbf{a} = [a_1 \ldots a_M]^T$ che minimizza un certa cifra di merito.

Essendo i sensori e i centri delle campane disposti ad intervalli regolari, ma con la distanza tra due campane maggiore rispetto a quella tra due sensori, si stabilisce che ogni N/M sensori, vi sia una "campana" abbinata al sensore stesso.

Ora poniamo il problema come ricerca del minimo di una funzione costo globale, si vuole minimizzare una cifra di merito:

$$F(\mathbf{a}) = \sum_{i=1}^{N} f_i(\mathbf{a}), \quad F : \mathbb{R}^M \to \mathbb{R} \tag{38}$$

dove $f_i(\mathbf{a})$ esprime il costo associato all'i-esimo sensore, in funzione del vettore di parametri $\mathbf{a}$. In ciascun istante f_i assumerà valori "piccoli" quando, in corrispondenza dell'i-esimo sensore, il valore stimato della mappa in quell'istante è vicino al valore misurato dal sensore. Nei paragrafi successivi si discuterà in maniera più approfondita di come sono fatte le f_i.

Il grafico in Fig. 5 illustra quanto spiegato finora: la linea rossa rappresenta il profilo "vero" che vogliamo ricostruire, a partire dall'informazione rumorosa di cui disponiamo (linea blu). A ciscuna linea verde infine corrisponde il centro di una campana.

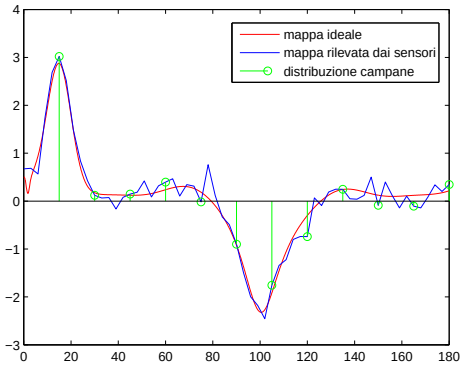

Fig. 5. Situazione di partenza del nostro esempio. La linea continua (rosso) è la mappa "vera" che vogliamo ricostruire come somma di funzioni elementari, a partire dall'osservazione rumorosa (blu). In verde sono indicati i centri delle campane che verranno utilizzate per la ricostruzione.

Per verificare quanto illustrato nella Sez. IV, si cercherà di risolvere il problema di mapping con un algoritmo di Newton-Raphson multivariabile centralizzato, dove si presume che il disturbo presenti degli outliers notevoli. Si suppone quindi, in base a quanto spiegato in precedenza, che l'utilizzo di funzioni

costo non quadratiche garantisca un funzionamento migliore rispetto alle classiche funzioni costo quadratiche.

A. Algoritmo di Newton-Raphson centralizzato per la ricostruzione della mappa

Come già anticipato, si utilizzeranno due diversi tipi di funzione costo. A partire dalla ricostruzione della mappa (37), presso ciascun agente viene calcolato un errore pesato secondo legge:

- quadratica, $q_i(\mathbf{a}) := \frac{1}{2}(y_i - G(x_i, \mathbf{a}))^2$;
- iperbolica, $h_i(\mathbf{a}) := \beta\sqrt{\frac{y_i - G(x_i,\mathbf{a})}{\alpha}^2 + 1}$.

y_i è la misura (rumorosa) acquisita dall'i-esimo sensore. L'algoritmo di Newton-Raphson multivariabile è realizzato con la nota formula

$$a(n+1) = a(n) - [H_F(a(n))]^{-1}\nabla F(a(n)) \tag{39}$$

dove $F(a(n))$ è la funzione costo globale (38) valutata all'istante n e $\nabla F(a(n))$, $H_F(a(n))$ sono rispettivamente gradiente e matrice Hessiana di $F(a(n))$.

Si vuole vedere come si sviluppa 39 nel nostro caso; ometteremo la dipendenza da n delle diverse espressioni trovate, sapendo che sono tutte riferite alla n-esima iterazione. Non specifichiamo il tipo di funzione costo usata, che denotiamo con $g(\cdot)$. Al posto di $g(x)$, $g'(x)$ e $g''(x)$ si sostituiscono i valori ottenuti con le espressioni (33), (34) e (35) nel caso iperbolico, o le equivalenti del caso quadratico.

Per calcolare in maniera esplicita $F(a)$, $\nabla F(a)$ e $H_F(a)$ si pone:

$$z_i := y_i - G(x_i, a) \tag{40}$$

$$\begin{aligned} F(a) &= g(z_1) + g(z_2) + \ldots + g(z_3) \\ &= g(y_1 - G(x_1, \mathbf{a})) + \ldots + g(y_N - G(x_N, \mathbf{a})) \end{aligned} \tag{41}$$

dove $G(x_i, \mathbf{a})$ è stata definita in (37). Per cui il gradiente $\nabla F(a(n))$ è dato da:

$$\begin{bmatrix} \frac{\partial F(a)}{\partial a_1} \\ \vdots \\ \frac{\partial F(a)}{\partial a_M} \end{bmatrix} = \begin{bmatrix} g'(z_1)\frac{\partial z_1}{\partial a_1} + \ldots + g'(z_N)\frac{\partial z_N}{\partial a_1} \\ \vdots \\ g'(z_1)\frac{\partial z_1}{\partial a_M} + \ldots + g'(z_N)\frac{\partial z_N}{\partial a_M} \end{bmatrix} \tag{42}$$

che sviluppando si può riscrivere:

$$\nabla F(a(n)) = -S \begin{bmatrix} g'(z_1) \\ \vdots \\ g'(z_N) \end{bmatrix} \tag{43}$$

avendo posto

$$S = \begin{bmatrix} e^{-(x_1 - w_1)^2} & \ldots & e^{-(x_N - w_1)^2} \\ e^{-(x_1 - w_2)^2} & \ldots & e^{-(x_N - w_2)^2} \\ \vdots & \ddots & \vdots \\ e^{-(x_1 - w_M)^2} & \ldots & e^{-(x_N - w_M)^2} \end{bmatrix} \tag{44}$$

con $S \in \mathbb{R}^{M \times N}$. Si omettono i calcoli per la matrice Hessiana, che risulta essere

$$H_F(a) = SG''S^T \tag{45}$$

dove abbiamo indicato con G'' la matrice diagonale :

$$G'' = \begin{bmatrix} g''(z_1) & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & g''(z_N) \end{bmatrix} \tag{46}$$

Si noti che, dal momento che la formula (39) prevede l'inversione della matrice Hessiana ad ogni iterazione, nel momento dell'implementazione si somma ad $H_F(a)$ una matrice diagonale δI_M, con δ piccolo, per evitare la divergenza nel caso la matrice Hessiana risulti singolare. L'algoritmo ad ogni iterazione associa a z_i il valore delle derivate prima e seconda della funzione costo $g(z)$, mentre la matrice A rimane invariata e dipende solo dalla disposizione spaziale di sensori e "campane". In particolare si vede come si potrebbe scegliere di utilizzare delle disposizioni di sensori irregolari senza che l'algoritmo si complichi, in quanto basterebbe cambiare i valori di x_i, w_j che determinano la matrice S.

Algorithm 3 Mapping con Newton-Raphson centralizzato.

(variabili)
a $\in \mathbb{R}^M$, $z \in \mathbb{R}^N$, $\nabla F(\cdot) \in \mathbb{R}^M$, $H_F(\cdot) \in \mathbb{R}^{M \times M}$
(parametri)
$y \in \mathbb{R}^N$ ▷ Vettore delle acquisizioni y_i $S \in \mathbb{R}^{M \times N}$ ▷ matrice S definita in (44)
(initializzazione)
a $= 0$
(algoritmo)
for $n = 1, 2, \ldots$ **do**
  $z(n) = y - S \cdot \mathbf{a}(n)$
  for $i = 1, 2, \ldots, N$ **do**
    $g(z_i(n)) = \ldots$
    $g'(z_i(n)) = \ldots$
    $g''(z_i(n)) = \ldots$
  end for
  $\nabla F(\mathbf{a}(n)) = -S \cdot g'(z(n))$
  $H_F(\mathbf{a}(n)) = \ldots$ ▷ Hessiana definita in (45)
  $\mathbf{a}(n+1) = \mathbf{a}(n) - [H_F(\mathbf{a}(n))]^{-1}\nabla F(\mathbf{a}(n))$
end for

Lo schema seguito per il mapping con un algoritmo di Newton-Raphson centralizzato è riassunto in Alg. 3 e risulta molto conveniente. Infatti, a partire dalle derivate prime e seconde delle funzioni costo presso ciascun agente, il nuovo vettore dei coefficienti $\mathbf{a}(n)$ viene calcolato in maniera rapida e con calcoli molto semplici, sfruttando il fatto che la matrice S rimane costante. Inoltre, supponendo che il calcolo delle $g(z_i)$, $g'(z_i)$ e $g''(z_i)$ avvenga presso ciascun agente, la complessità totale dei calcoli effettuati dall'intelligenza centrale dipende unicamente dal numero di campane utilizzate, ovvero la dimensione M della matrice da invertire.

La differenza tra il metodo a funzioni quadratiche e quello a funzioni iperboliche sta tutta nelle diverse formule per i vettori $F(a(n))$ e $\nabla F(a(n))$ e per la matrice $H_F(a(n))$, che si ottengono scegliendo una funzione costo $g(z)$ dell'uno o dell'altro tipo. In Fig. 6 sono presentati i risultati dell'algoritmo centralizzato, con i due diversi tipi di funzione costo, a partire dall'osservazione rumorosa in Fig. 5.

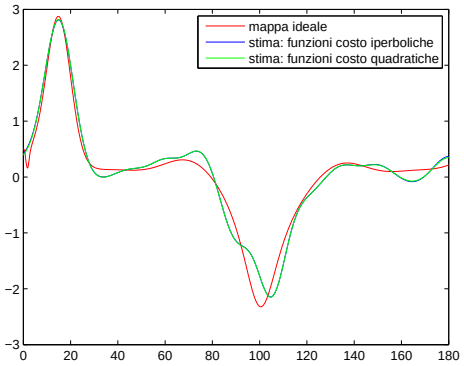

Fig. 6. Ricostruzione della mappa con l'Alg. 3, in assenza di outliers. Confronto tra funzioni costo iperboliche e quadratiche.

I due risultati che si ottengono sono sostanzialmente indistinguibili, quindi possiamo affermare che, nelle condizioni in cui è stato fatto girare l'algoritmo, è equivalente ai fini della qualità della ricostruzione della mappa il tipo di funzione costo adottato. Naturalmente il metodo quadratico utilizza calcoli più semplici di quello iperbolico e perciò sarebbe preferibile.

Il profilo della mappa "vera" che vogliamo ricostruire è stato generato arbitrariamente come somma di diverse funzioni. La prima grossa protuberanza (di ascissa circa 15) è stata generata con una funzione esponenziale dello stesso tipo di quelle utilizzate per la ricostruzione della mapppa, la seconda invece (negativa, di ascissa circa 100) è dovuta ad una funzione razionale[5]. Si nota quindi come l'algoritmo riesca a riprodurre con molta precisione una mappa se questa è stata generata con lo stesso tipo di funzioni che si utilizzano per ricrearla, mentre invece approssima in maniera peggiore funzioni diverse da quelle utilizzate per la ricostruzione.

I due metodi producono risultati completamente diversi invece, se al normale rumore gaussiano di acquisizione, si sommano delle distorsioni molto elevate in corrispondenza all'acquisizione di un piccolo numero di sensori, che è il caso della Fig. 7.

In Fig. 8 si vedono i risultati dell'algoritmo di Newton-Raphson centralizzato con le funzioni costo locali quadratiche e iperboliche.

Nel caso di funzioni costo quadratiche l'outlier negativo (corrispondente ad una ascissa circa pari a 80) rovina la qualità della ricostruzione della mappa, infatti viene introdotta una nuova depressione che porta a ricostruire un profilo totalmente inesatto. Al contrario, con le funzioni costo iperboliche la ricostruzione è molto meno influenzata dall'outlier, e l'errore che si commette rispetto al profilo vero è accettabile.

[5]Le espressioni esplicite delle due funzioni sono quelle riportate all'inizio della Sez. V

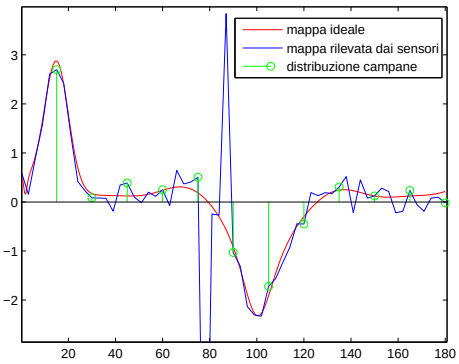

Fig. 7. Situazione di partenza in presenza di outliers. Il picco rumorose negativo, che esce dalla figura, assume un valore pari a circa -8.

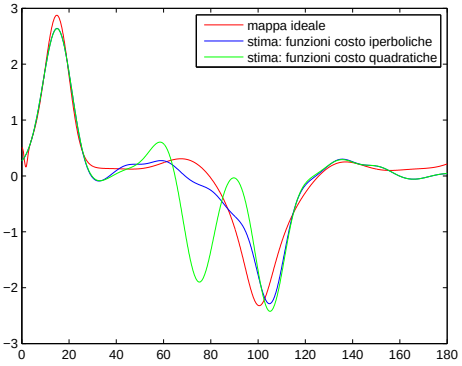

Fig. 8. Ricostruzione della mappa con l'Alg. 3 ed in presenza di outliers. Confronto tra funzioni costo iperboliche e quadratiche.

B. Algoritmo di Newton-Raphson distribuito per la ricostruzione della mappa

Arrivati a questo punto, abbiamo appurato che le funzioni costo a crescita lineare permettono di neutralizzare in parte l'effetto negativo degli outliers che compaiono nelle acquisizioni di certi segnali, nel momento in cui si procede alla loro interpolazione.

L'esempio che abbiamo illustrato mostra come si possa ricostruire una mappa utilizzando un algoritmo di Newton-Raphson multivariabile classico, che prevede la possibilità di acquisire ad ogni iterazione l'informazione proveniente da ciascun sensore per elaborare i calcoli (in gran parte derivate) presenti nell'algoritmo.

Ora, vogliamo riproporre il caso di funzioni costo iperboliche, nel quale tuttavia ad una struttura centralizzata sostituiamo una logica distribuita. In particolare utilizzeremo l'Alg. 4, estensione al caso multivariabile dell'Alg. 1.

Algorithm 4 Ottimizzazione distribuita

(variabili)

1: $x_i(n)$, $y_i(n)$, $g_i(n) \in \mathbb{R}^{N\times N}$ for $i = 1, \dots, M$ and $k = 1, 2, \dots$

(parametri)

2: $P \in \mathbb{R}^{M\times M}$, consensus matrix

3: $\varepsilon \in (0, 1)$

(inizializzazione)

4: **for** $i = 1$ to M **do**

5: $y_i(0) = g_i(-1) = 0$

6: $Z_i(0) = H_i(-1) = 0$

7: $x_i(0) = 0$

8: **end for**

(algoritmo)

9: **for** $n = 1, 2, \dots$ **do**

(aggiornamenti locali)

10: **for** $i = 1$ to M **do**

11: $g_i(n) = \nabla^2 f_i(x_i(n))x_i(n) - \nabla f_i(x_i(n))$

12: $H_i(n) = \nabla^2 f_i(x_i(n))$

13: $y_i(n) = y_i(n-1) + g_i(n-1) - g_i(n-2)$

14: $Z_i(n) = Z_i(n-1) + H_i(n-1) - H_i(n-2)$

15: **end for**

("average consensus" multidimensionale)

16: $Y(n) = (P \otimes I_N)Y(n)$

17: $\mathbf{Z}(n) = (P \otimes I_N)\mathbf{Z}(n)$

(aggiornamenti locali)

18: **for** $i = 1$ to M **do**

19: $x_i(n) = (1-\varepsilon)x_i(n-1) + \varepsilon(Z_i(n))^{-1}y_i(n)$

20: **end for**

21: **end for**

In maniera sintetica, si può affermare che la grande differenza tra questo metodo e quello precedente è che in questo caso la stima della mappa non è più univoca ma ci saranno tante stime quanti sono i sensori, che, grazie allo scambio di informazioni tra di loro, convergono asintoticamente ad un'unica stima, in maniera del tutto analoga a quanto visto in II nel caso scalare.

Ad ogni iterazione dell'algoritmo, che ricordiamo avviene in ciascun sensore contemporaneamente, possiamo distinguere tre fasi diverse:

- Fase 1: aggiornamento della distanza tra la propria misura e quella stimata dall'agente, successiva elaborazione dei vettori Z e Y;
- Fase 2: scambio di informazione tra gli agenti e i propri vicini;
- Fase 3: aggiornamento dei vettori di coefficienti sulla base della nuova informazione.

1) Fase 1: All'inizio dell'n-esimo ciclo, l'i-esimo agente ha un vettore di parametri $\mathbf{a}^{(i)}(n)$ cui corrisponde una determinata funzione discreta $G(\cdot, \mathbf{a}^{(i)}(n))$, e ne calcola il valore nel proprio punto (x_i) confrontandolo con quello misurato y_i, per ottenere così l'errore locale di approssimazione. Per l'i-esimo agente, si avrà così:

$$z_i(\mathbf{a}^{(i)}(n)) = y_i - G(x_i, \mathbf{a}^{(i)}(n)) \tag{47}$$

Si noti come la (40) si differenzi dalla (47) in quanto presso ciascun sensore si avrà una diversa ricostruzione della mappa $G(\cdot, \mathbf{a}^{(i)})$ che dipende dal vettore $\mathbf{a}^{(i)}$ a disposizione dell'i-esimo sensore in quell'istante.

A partire da questo dato, ciascun agente aggiorna i vettori H_i e Z_i, analogo multivariabile degli elementi dei vettori $\mathbf{z}$ e $\mathbf{y}$ in 12.

L'aggiornamento avviene in maniera analoga a quanto visto in II, solo che in questo caso le quantità $g_i(x_i(n))$ e $h_i(x_i(n))$ non sono più scalari bensì vettori e matrici, di conseguenza nel loro calcolo intervengono gradienti ed Hessiane al posto di derivate prime e seconde.

Volendo sviluppare le espressioni per $g_i(x_i(n))$ e $H_i(x_i(n))$, si riscrive la (47) in maniera esplicita come:

$$y_i - a_1^{(i)} e^{-(x_i - w_1)^2} - \dots - a_M^{(i)} e^{-(x_i - w_M)^2} \tag{48}$$

A partire da questa forma risulta immediato calcolare il gradiente della i-esima funzione costo $f(z_i)$[6].

$$\nabla f(z_i) = f'(z_i) \begin{bmatrix} \frac{\partial z_i}{\partial a_1} \\ \vdots \\ \frac{\partial z_i}{\partial a_M} \end{bmatrix} = -f'(z_i) S_i \tag{49}$$

avendo posto:

$$S_i = \begin{bmatrix} e^{-(x_i - w_1)^2} \\ \vdots \\ e^{-(x_i - w_M)^2} \end{bmatrix} \tag{50}$$

Infine la j-esima riga della matrice Hessiana $H_i(z_i) = \nabla^2 f_i(z_i)$ è il gradiente (trasposto) del j-esimo elemento di $\nabla f(z_i)$, per cui si può calcolare l'elemento in posizione (j, k) di $H_i(z_i)$ come

$$\begin{aligned} \frac{\partial^2 f(z_i)}{\partial a_j \partial a_k} &= f''(z_i)\frac{\partial z_i}{\partial a_j}\frac{\partial z_i}{\partial a_k} + f'(z_i)\frac{\partial^2 z_i}{\partial a_j \partial a_k} \\ &= f''(z_i)\frac{\partial z_i}{\partial a_k}\frac{\partial z_i}{\partial a_j} \\ &= f''(z_i)e^{-(x_i - w_k)^2 - (x_i - w_j)^2} = f''(z_i) S_i S_i^T \end{aligned} \tag{51}$$

Anche nel caso distribuito, così come si è trovato nel caso centralizzato, ad ogni iterazione vengono calcolati unicamente i nuovi valori delle derivate prime e seconde corrispondenti alle funzioni costo locali. A partire da questi vengono calcolati i gradienti e le matrici hessiane, sfruttando l'informazione memorizzata in ciascun agente, ovvero il vettore S_i.

[6]Indichiamo le funzioni costo relative a ciascun sensore con $f(z_i)$ anziché $g(z_i)$, per non fare confusione con il vettore $g_i(x_i(n))$, aggiornato nella riga 11 dell'Alg. 4, che viene calcolato proprio a partire dalla funzione costo locale $f(z_i)$).

Una volta calcolati g_i e H_i ciascun agente aggiorna i suoi valori di y_i e Z_i (righe 13 e 14 dell'algoritmo) ed è pronto alla fase successiva.

2) Fase 2: Lo scambio di informazione tra i diversi agenti (che nel nostro caso sono i sensori stessi) avviene univocamente con gli agenti vicini, tuttavia per simulare in Matlab il passo di consenso multivariabile, non si utilizza una matrice circolare di dimensione $N \times N$ del tipo:

$$P = \begin{bmatrix} 0.5 & 0.25 & 0 & \dots & 0 & 0.25 \\ 0.25 & 0.5 & 0.25 & \dots & 0 & 0 \\ 0 & 0.25 & 0.5 & \dots & 0 & 0 \\ \dots & & & & & \\ 0.25 & 0 & \dots & 0 & 0.25 & 0.5 \end{bmatrix} \tag{52}$$

bensì una matrice di dimensione $NM \times NM$, che si ottiene operando un prodotto di Kronecker tra la matrice P sopracitata e una matrice identità di dimensione M.

Il prodotto di Kronecker tra due matrici $A \in \mathbb{R}^{m \times n}$ e $B \in \mathbb{R}^{p \times q}$ è dato da

$$K = A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix} \tag{53}$$

Nel nostro caso rappresenta con un'unica operazione matriciale tutti gli scambi di informazione riguardanti i vettori dei coefficienti che avvengono tra i diversi agenti.

3) Fase 3: Una volta scambiata l'informazione con gli agenti vicini, ciascun agente in maniera autonoma riaggiorna il suo vettore di coefficienti $\mathbf{a}_i$, come descritto nell'ultima riga dell'Alg. 3. Anche in questo caso, come nel caso centralizzato, l'inversione della matrice Z_i può comportare la divergenza dell'algoritmo, nel caso la matrice risulti singolare. Perciò, nell'implementare l'algoritmo, si somma sistematicamente a Z_i una matrice diagonale ottenuta moltiplicando la matrice unitaria per un fattore δ molto piccolo (in questa simulazione si è utilizzato $\delta = 5 \cdot 10^{-4}$).

In quest'ultima fase ciascun agente deve anche memorizzare i valori di $g_i(k-1)$ e $H_i(k-2)$ che andranno sottratti all'iterazione successiva per l'aggiornamento di y_i e Z_i (righe 13 e 14).

Per quanto riguarda la pesantezza dell'algoritmo presso ciascun agente, valgono le stesse considerazioni fatte in V, ovvero la quantità di calcolo da effettuarsi è principalmente dovuta all'inversione della matrice Z_i di dimensione $M \times M$ in riga 19. Si potrebbe notevolmente diminuire questa complessità, riformulando l'algoritmo in modo che ciascun agente si occupi unicamente del calcolo dei coefficienti a_j che contribuiscono effettivamente alla ricostruzione della mappa presso il proprio sensore.

Per applicare il metodo proposto ad una mappa ottenuta a partire da osservazioni rumorose ed in presenza di outliers, utilizziamo le funzioni costo iperboliche illustrate nella Sez. IV. Nelle Fig. 9 e 10 si può osservare l'evoluzione della mappa stimata dal primo sensore/agente. La parte iniziale e finale della mappa viene ricostruita molto più velocemente di quella centrale, perché si è supposto che la comunicazione tra gli agenti sia circolare, e di conseguenza il primo sensore riesce ad ottenere informazione dagli ultimi sensori prima rispetto a quella proveniente dagli agenti centrali.

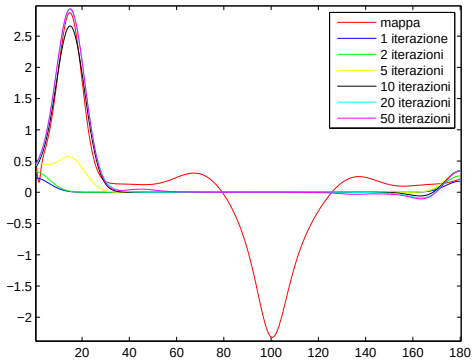

Fig. 9. Stima della mappa disponibile presso il primo sensore ($x = 0$). Andamento durante le prime 100 iterazioni.

In Fig. 10 si può notare come la stima dell'intera mappa raggiunga una condizione di regime dopo $5 \div 600$ iterazioni.

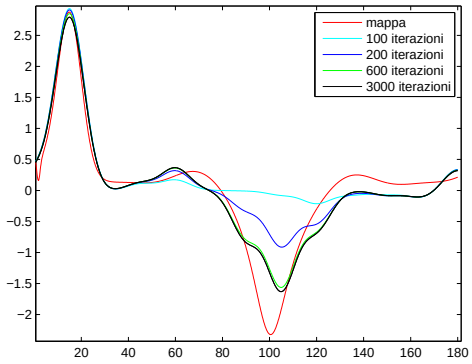

Fig. 10. Stima della mappa disponibile presso il primo sensore ($x = 0$). Andamento a partire dalla centesima iterazione e a regime.

Nelle Fig. 11 e 12 invece si osserva l'andamento nel tempo della mappa disponibile presso il 30-esimo sensore (posizionato a metà mappa, dal momento che i sensori sono totalmente 61).

Come si poteva prevedere, a differenza di quanto succede per il primo sensore, il sensore più centrale ricostruisce velocemente la porzione di mappa più vicina a lui, mentre bisogna aspettare sempre circa $5 \div 600$ iterazioni perché venga

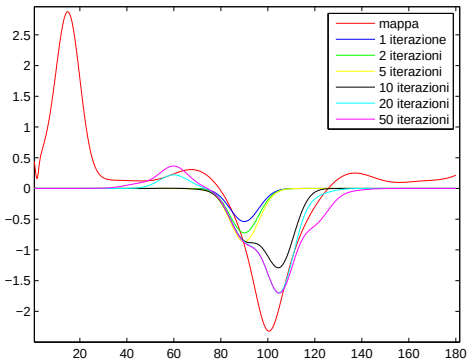

Fig. 11. Stima della mappa disponibile presso il trentesimo sensore ($x = 87$). Andamento durante le prime 100 iterazioni.

identificato il primo picco, e si possa considerare il consenso raggiunto.

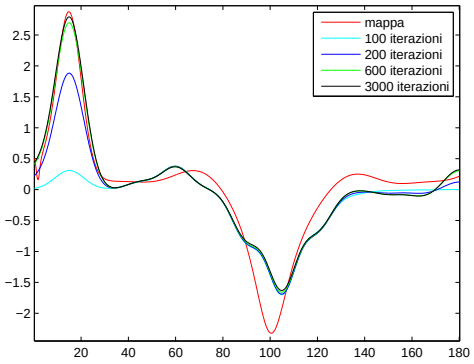

Fig. 12. Stima della mappa disponibile presso il trentesimo sensore ($x = 87$). Andamento a partire dalla centesima iterazione e a regime.

Anche qui, come nel caso dell'algoritmo di Newton-Raphson centralizzato visto in Sez. V-A, la qualità della ricostruzione della mappa è nettamente migliore nei punti in cui la funzione analitica con cui è stata ottenuta la mappa vera è costituita da particolari funzioni esponenziali come quelle usate usate per l'identificazione.

C. Considerazioni sul mapping tramite algoritmi di Newton-Raphson

In conclusione si può affermare che sono stati raggiunti i risultati previsti: si è dimostrato che in presenza di outliers l'utilizzo di funzioni costo a crescita lineare anziché quadratica permette di limitare l'effetto negativo degli outliers nelle misure.

Inoltre si è riusciti ad implementare un'applicazione robusta dell'Alg. 1, in una sua variante multidimensionale, in un caso pratico connotato dalla presenza di forte rumore nell'acquisizione dei dati.

VI. Studio sul fattore di scala temporale ϵ

A. Ruolo del fattore ε

Nella Sez. II è stato presentato l'Alg. 1[7] che utilizza una matrice di consensus costante e richiede la definizione del valore del parametro ε. Esso gioca un ruolo fondamentale sia per quanto riguarda la stabilità che per la velocità di convergenza. Il Teorema 11.4 in [13] stabilisce l'esistenza di un valore critico ε^* tale che per ogni $\varepsilon < \varepsilon^*$ è garantita la stabilità esponenziale del sistema 17. Quindi è sufficiente individuare tale valore critico per poter far girare l'algoritmo correttamente adottando un opportuno valore del parametro e mantenendolo costante per tutta l'esecuzione del programma. Sfortunatamente il teorema precedente non suggerisce alcun metodo per individuare il valore critico, inoltre esso cambia al variare delle funzioni costo e delle condizioni iniziali. Quindi non c'è modo di sapere, a priori, il valore ideale da assegnare a tale parametro.

Che cosa succede se si scelglie un valore errato? Se esso risulta maggiore di ε^* c'è la possibilità che l'andamento delle variabili di stato x diverga e che non si riesca ad individuare il punto di mnimo globale. Se, invece, si sceglie un valore eccessivamente basso, ad ogni iterazione dell'Alg. 1 le variabili di stato variano lentamente e aumentano sensibilmente i cicli necessari per convergere al minimo.

Anziché cercare il valore critico ε^* prima di eseguire l'algoritmo e poi adottarlo come costante per tutta l'esecuzione, si può escogitare un metodo dinamico per il tuning del parametro ε che si occupi di modificarne il valore in base alle prestazioni correnti. Per fare ciò bisogna studiare un criterio che permetta di capire se il valore attuale è eccessivamente alto o, al contrario, troppo basso. Un approccio di questo tipo, non solo permetterebbe di aggirare il problema di individuare il valore ciritico ε^*, ma potrebbe migliorare le prestazioni dell'algoritmo riducendo il numero di iterazioni totali grazie all'adozione un valore di ε variabile invece che costante[8].

In seguito verrà illustrato l'approccio che è stato adottato per il tuning del parametro ε e i passaggi che hanno portato a tale soluzione. In primo luogo verrà esposto lo studio di una modifica all'Alg. 1[9] che consenta l'*autotuning* di un unico parametro ε comune per tutti gli agenti (caso centralizzato), successivamente verrà applicata la stessa strategia all'Alg. 1 modificato con la versione asincrona presentata in Alg. 2, ma con un parametro diverso associato ad ogni agente (caso distribuito).

B. Controllo adattativo, caso centralizzato

Studiando il comportamento temporale delle variabili di stato dato un valore fissato del parametro ε, si può distinguere una prima fase in cui gli stati divergono (anche se partono dagli

[7]In tutti gli algoritmi che seguono è stato tralasciata l'indicazione di M per semplificare la notazione e si è sempre assunto M=1

[8]Per come è stato costruito l'algoritmo, adottare valori di ε arbitrariamente grandi in un primo momento non pregudica la convergenza, a patto di rudurli opportunamente in seguito.

[9]con matrice di *consensus* P costante.

stessi valori iniziali), mentre successivamente essi convergono verso un valore comune che si avvicina progressivamente al punto di minimo globale[10]. Facendo variare il parametro ε (mantenendolo comunque inferiore a ε^*) si nota che la fase di divergenza iniziale risulta più ampia (in termini di distanza dei valori degli stati dal punto di minimo) quando ε è elevato, mentre quando ε è molto basso si ha che la seconda fase è caratterizzata da una convergenza asintotica marcatamente più lenta.

Algorithm 5 Aumento di ε

if $\varepsilon < 0.5$ **then**
  $\varepsilon = 1.5\varepsilon$
else
  $\varepsilon = \varepsilon + (1-\varepsilon)/2.5$
end if

La strategia più intuitiva che si possa pensare è quella di un controllo adattativo che parta da un valore qualsiasi[11] del parametro ε e lo modifichi diminuendolo quando lo stato $\mathbf{x}$ si trova distante dal punto di minimimo e aumentandolo quando lo stato si trova nelle sue vicinanze, in modo da privilegiare la fase di *consensus* rispetto a quella di *Newton-Raphson* nel primo caso quando gli stati tendono a seguire traiettorie diverse e divergenti e viceversa nel secondo quando gli stati convergono ad un valore comune.

Il problema è che la distanza dal punto di minimo non è quantificabile dato che tale punto è, ovviamente, ignoto a priori. Ne segue che l'unico strumento che si ha a disposizione per capire quanto distanti ci si trovi dal punto di minimo è l'andamento temporale delle variabili di stato. Una soluzione può essere quella di monitorare, ad ogni iterazione dell'algoritmo, l'incremento $\delta(k) = |x(k) - x(k-1)|$ delle variabili di stato rispetto al valore precedente.

L'idea di base è che se tale incremento cresce col tempo, cioè se $\delta(k) > \delta(k-1)$, si può presumere di essere in fase divergente, ovvero che il valore corrente del paramentro ε risulta troppo elevato, mentre se δ decresce significa che si è probabilmente giunti in fase di convergenza asintotica al punto di minimo e si può aumentare ε per risparmiare iterazioni[12].

Ora è chiaro come variare il paramentro ε, ma non è stato ancora definito di quanto esso deve essere modificato. Il procedimento che è stato scelto è quello di dimezzare il valore di ε ripetutamente quando δ cresce (fase divergente) e di raddoppiarlo quando δ diminuisce. Le motivazioni di questa scelta risiedono nel fatto che è una soluzione estremamente semplice e l'andamento esponenziale che la caratterizza consente di raggiungere rapidamente il valore ideale del parametro ε e di seguirne le variazioni.

Algorithm 6 Versione base dell'algoritmo di tuning del parametro ε

$\boldsymbol{\delta}(k-1) = \left| \frac{\mathbf{y}(k-1)}{\mathbf{z}(k-1)} - \frac{\mathbf{y}(k-2)}{\mathbf{z}(k-2)} \right|$
$\boldsymbol{\delta}(k) = \left| \frac{\mathbf{y}(k)}{\mathbf{z}(k)} - \frac{\mathbf{y}(k-1)}{\mathbf{z}(k-1)} \right|$
if $\sum_{i=1}^{N} \delta_i(k-1) > \sum_{i=1}^{N} \delta_i(k)$ **then** ▷ aumenta ε
  if $\varepsilon < 0.5$ **then**
    $\varepsilon = 1.5\varepsilon$
  else
    $\varepsilon = \varepsilon + (1-\varepsilon)/2.5$
  end if
else ▷ riduci ε
  $\varepsilon = \varepsilon/2$
end if

Quello abbozzato nel paragrafo precedente rappresenta solo un grezzo tentativo iniziale e necessita di essere affinato ulteriormente per ottenere buoni risultati. In particolare sono importanti i seguenti accorgimenti. Innanzitutto, dato che evitare la divergenza è più importante che velocizzare l'algoritmo, anziché raddoppiare ε si è deciso di moltiplicarlo per un fattore 1.5[13]. In secondo luogo ε deve appartenere all'intervallo $[0, 1]$, quindi non può essere aumentato indefinitivamente; quando ε raggiunge il valore 0.5 la procedura di aumento diventa meno aggressiva portando il parametro ad avvicinarsi asintoticamente al valore 1 come esposto in Alg. 5.

Un ulteriore dettaglio da specificare nella versione centralizzata dell'algoritmo, ovvero quella che presenta un parametro ε comune per tutti gli agenti, è come trattare il fatto che nonostante lo stato associato ad ogni agente cambi in modo differente da tutti gli altri[14], la sua variazione deve essere ricondotta all'unico ε. Esisteranno quindi N incrementi

$$\delta_i(k) = |x_i(k) - x_i(k-1)|, \quad i = 1, ..., N \tag{54}$$

dove N è il numero di agenti e, ad ogni iterazione, l'aggiornamento di ε verrà effettuato in base alla loro media aritmetica[15]. Questa soluzione ha il duplice vantaggio di prendere in considerazione gli incrementi di tutti gli stati in una volta sola e che brusche variazioni di uno o pochi stati hanno ripercussioni meno negative in quanto vengono, in un certo senso, bilanciate da quelle più moderate degli altri stati.

[10] Si vedano a tale proposito gli esempi riportati in [1].

[11] In tutti gli esempi seguenti il valore di partenza assegnato al parametro ε è 0.001

[12] Si noti che quando δ cresce non è necessariamente vero che i valori delle variabili di stato si stiano allontanando dal punto di minimo globale, al contrario potrebbero effettivamente convergervi in veloce avvicinamento (ad esempio se le condizioni di partenza dovessero prevedere valori iniziali molto distanti dal punto di minimo). In ogni caso, comunque, diminuire il valore di ε non avrebbe effetti negativi per la convergenza se non quello di diminuirne la velocità. Se si verificasse ciò, si ridurrebbero di conseguenza gli incrementi delle variabili di stato e si ricadrebbe nel secondo caso, per il quale è previsto l'aumento del valore di ε e la velocità di avvicinamento tornerebbe ad aumentare.

[13] Ottenuto con acluni test su funzioni di prova. Si sottolinea che il valore esatto di questo fattore non è di fondamentale importanza, quello che conta è che la procedura di aumento di ε sia leggermente più lenta di quella di dimezzamento.

[14] Pur rimanendone influenzato grazie alla fase di *consensus*.

[15] Si noti che in realtà non è necessario calcolare la media, ma è sufficiente valutare se la semplice somma dei $\delta_i(k)$ è maggiore o minore di quella dei $\delta_i(k-1)$.

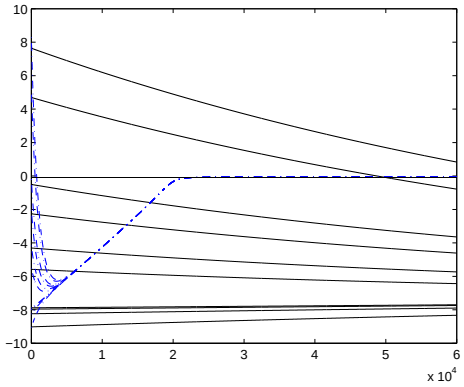

Fig. 13. Evoluzioni temporali degli stati $x_i(k), i = 1, ..., N$ per $N = 10$ e coefficienti come in (56) ottenute con ε costante pari a 0.001, di poco inferiore a ε^*, (in blu a tratti) e con ε variabile secondo l'Alg. 6 (in nero). Risulta evidente che tale strategia non è sufficientemente veloce. In questo e negli esempi successivi, il punto di minimo globale si trova in $x = -0.05$.

Ora è possibile stendere un primo listato completo del programma. Il codice mostrato in Alg. 6[16] contiene la procedura di tuning del parametro e deve essere considerato racchiuso all'interno del ciclo **for** presente in Alg. 1 alla riga 6.

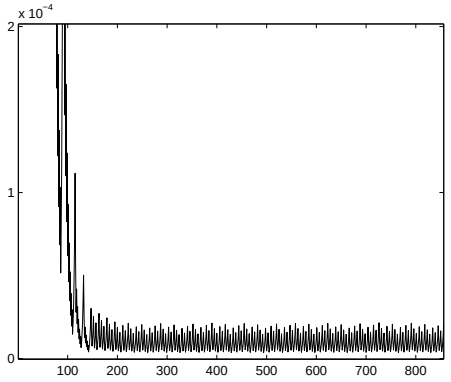

Fig. 14. Andamento temporale del parametro $\varepsilon(k)$ secondo l'Alg. 6. Si notano ripetute oscillazioni dovute al fatto che una variazione del parametro si ripercuote sull'evoluzione degli stati che, a sua volta, disturba la successiva variazione del parametro.

C. Miglioramento dell'algoritmo, introduzione del periodo τ

Facendo girare l'Alg. 6 si ottengono buoni risultati nel calcolo del minimo di funzioni costo simili a quelle presentate come esempio in [1], ma se si scelgono come funzioni costo somme di esponenziali con esponenti e coefficienti più elevati in modulo del tipo:

$$f_i(x) = c_i e^{a_i x} + d_i e^{-b_i x}, \quad i = 1, ..., N \tag{55}$$

[16]Le linee di frazione che compaiono nelle righe di codice 1 e 2 indicano la divisione di Hadamard, ovvero la divisione tra vettori elemento per elemento e il segno di modulo indica il vettore dei moduli. Analogo discorso vale per gli algoritmi che seguono

$$a_i, b_i \sim \mathcal{U}[0, 3], \quad c_i, d_i \sim \mathcal{U}[0, 1], \tag{56}$$

cominciano ad emergere alcuni problemi. In particolare la velocità di convergenza al punto di minimo risulta estremamente lenta (si veda Fig. 13). In sostanza le prestazioni non sono assolutamente paragonabili a quelle che si otterebbero utilizzando un valore costante per il parametro ε di poco inferiore a ε^*.

Per cercare di capire la causa di questo comportamento si

Algorithm 7 Algoritmo di tuning del parametro ε con periodo di aggiornamento τ

1: $\boldsymbol{\delta}(h-\tau) = \left| \frac{\mathbf{y}(h-\tau)}{\mathbf{z}(h-\tau)} - \frac{\mathbf{y}(h-\tau-\bar{\tau})}{\mathbf{z}(h-\tau-\bar{\tau})} \right|$
2: $\boldsymbol{\delta}(h) = \left| \frac{\mathbf{y}(h)}{\mathbf{z}(h)} - \frac{\mathbf{y}(h-\tau)}{\mathbf{z}(h-\tau)} \right|$
3: **if** $\sum_{i=1}^{N} \delta_i(h-\tau) > \sum_{i=1}^{N} \delta_i(h)$ **then** ▷ aumenta ε
4: **if** $\varepsilon < 0.5$ **then**
5: $\varepsilon = 1.5\varepsilon$
6: **else**
7: $\varepsilon = \varepsilon + (1 - \varepsilon)/2.5$
8: **end if**
9: $\tau = \tau/2$ ▷ e riduci il periodo
10: **if** $\tau < 1$ **then**
11: $\tau = 1$
12: **end if**
13: **else** ▷ riduci ε
14: $\varepsilon = \varepsilon/2$
15: $\tau = \tau * 2$ ▷ e aumenta il periodo
16: **end if**

può osservare l'andamento nel tempo del parametro ε. Esso è caratterizzato dalla presenza di oscillazioni ripetute in cui il valore di ε viene alternativamente alzato e diminuito, come se si passasse continuamente dalla fase divergente a quella convergente e viceversa (si veda Fig. 14). Pure nell'andamento degli incrementi delle variabili di stato sono state riscontrate delle oscillazioni (anche se molto meno marcate perché i bassi valori assunti dal parametro ε si traducono in piccole variazioni di stato). A cosa sono dovute?

La risposta sta nel fatto che nel sistema che stiamo studiando, lo stato corrente dipende dal valore di ε e con il criterio di tuning proposto è stata chiusa la retroazione tra lo stato e il parametro stesso. In questo modo il sistema diventa molto sensibile alle variazioni dello stato che si ripercuotono negativamente su ε. Le oscillazioni possono essere viste come un disturbo indotto dall'azione di tuning del parametro ε che si inserisce nel *loop* e bisogna adottare un accorgimento per renderlo ininfluente.

Al posto di valutare gli incrementi δ_i e aggiornare il valore di ε ad ogni iterazione dell'algoritmo, si possono eseguire tali operazioni meno frequentemente. In particolare si può au-

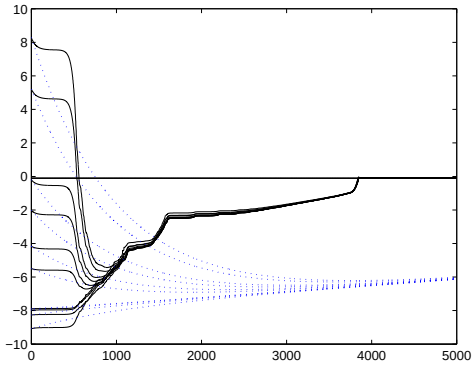

Fig. 15. Evoluzioni temporali degli stati $x_i(k), i = 1, ..., N$ per $N = 10$ e coefficienti come in 56 ottenute con ε costante pari a 0.001, di poco inferiore a ε^*, (in blu a tratti) e con ε variabile secondo l'Alg. 7 (in nero). A differenza dell'esempio precedente, questa procdura non solo permette di partire da un valore iniziale di ε qualsiasi, ma si rivela più veloce di quella con parametro costante e circa paria a ε.

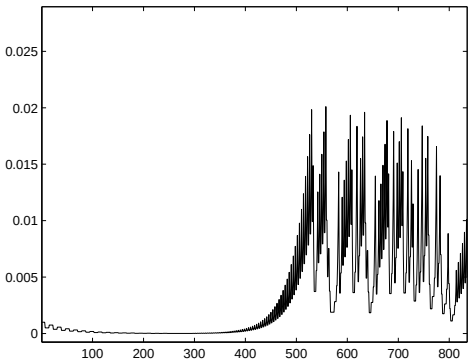

Fig. 16. Andamento temporale del parametro $\varepsilon(k)$ secondo l'Alg. 7. Le oscillazioni sono state notevolmente ridotte dall'introduzione del periodo τ, permettendo un aumento delle prestazioni in termini di numero totale di iterazioni richieste.

mentare il periodo di campionamento $\tau \in \mathbb{N}$ degli incrementi:

$$\delta_i(h) = |x_i(h) - x_i(h - \tau)|, \qquad h = \tau k, \quad k = 1, 2, ... \tag{57}$$

variando il parametro ε solamente ogni τ iterazioni in modo da mascherare le oscillazioni dovute al disturbo e da catturare solo la corretta evoluzione di stato. Il ritardo che viene introdotto, infatti, permette di modificare ε in base ai movimenti di stato e non alle variazioni del parametro stesso. Purtroppo questa elegante soluzione ha un prezzo: il problema ora diventa determinare il periodo ideale τ.

Se si sceglie un τ troppo piccolo, il sistema continua ad essere troppo sensibile al disturbo e le prestazioni non migliorano molto. Al contrario se il periodo è troppo lungo, c'è il rischio di restare bloccati per troppi cicli su un valore di ε sbagliato (eccessivo, che causerebbe divergenza, o troppo basso, che porterebbe ad una convergenza lenta). Diventa necessario studiare un automatismo che permetta di determinare quale sia il valore di τ da adottare.

Analogamente a quanto fatto per il tuning del parametro ε, anche in questo caso si è deciso di adottare una procedura che prevede di raddoppiare o dimezzare il periodo τ ripetutamente secondo le seguenti condizioni. Il valore di τ viene raddoppiato ogni volta che si decide di ridurre ε, mentre viene dimezzato quando si aumenta ε. L'idea alla base di questa soluzione è che quando si può alzare il parametro, è opportuno aggiornarlo il più rapidamente possibile perché così si aumenta la velocità di convergenza e si controlla con maggior frequenza che il suo valore non sia troppo alto, evitando di divergere per un periodo troppo lungo. Il listato è illustrato in Alg. 7[17]. Con questo accorgimento, le prestazioni migliorano notevolmente (si veda Fig. 15).

[17]In questo e negli algoritmi seguenti, $\bar{\tau}$ indica la precedente lunghezza del periodo, ovvero prima che venisse aggiornata in τ. Il valore iniziale assegnato a τ è 8.

D. Perfezionamento delle prestazioni

L'introduzione del periodo τ ha permesso di ridurre le oscillazioni del parametro ε e quindi di diminuire il numero di cicli totali richiesti per convergere al punto di minimo. Osservando l'andamento di ε nel tempo, però, si può notare che sono ancora presenti alcune oscillazioni. Esse possono essere interpretate nel seguente modo: l'algoritmo tenta di raggiungere un determinato valore di ε, ma, dato che può procedere solamente moltiplicando il valore corrente per 1.5 o dividendolo per 2, ad ogni iterazione esso assume valori in un intorno di quello cercato senza raggiungerlo. Per evitare che questa situazione altalenante sprechi troppi cicli, è sufficiente adottare un procedimento dicotomico per avvicinarsi con precisione al valore richiesto. Se ε deve essere aumentato dopo essere stato diminuito (o viceversa) ad esso viene assegnato un valore intermedio tra quello corrente e quello precedente, il codice aggiornato è esposto in Alg. 8, mentre in Fig. 17 è mostrato un dettaglio dell'andamento del parametro ε secondo la strategia di tuning appena esposta.

Con quest'ultimo accorgimento, i cicli macchina richiesti per raggiungere il punto di minimo sono ulteriormente ridotti (si veda Fig. 18).

Quello che è stato realizzato, in conclusione, è un algoritmo in grado di trovare il punto di minimo della funzione costo partendo da un valore qualsiasi del parametro ε, ma ottenendo buone prestazioni anche quando esso è molto diverso da ε^*, addirittura migliori della versione con parametro costante.

E. Controllo adattativo, caso distribuito

Quello illustrato fino ad ora, però, è un algoritmo in cui il parametro ε è condiviso da tutti gli agenti. In questa sezione, verrà mostrata una versione distribuita del criterio di tuning e verrà applicata all'Alg. 1 modificato con la versione asincrona presentata in Alg. 2, con matrice di *consensus* P variabile. Per implementare una versione distribuita dello stesso, è necessario assegnare un diverso ε_i ad ogni agente e riadattarne la procedura di aggiornamento.

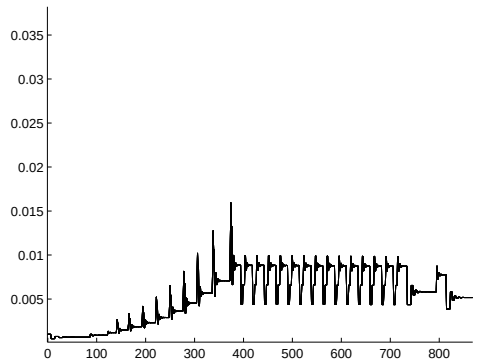

Fig. 17. Andamento temporale del parametro $\varepsilon(k)$ secondo l'Alg. 8. Le oscillazioni sono pressocché sparite e si può notare molto bene come il parametro tenda ad attestarsi dicotomicamente attorno a determinati valori nei vari istanti di tempo.

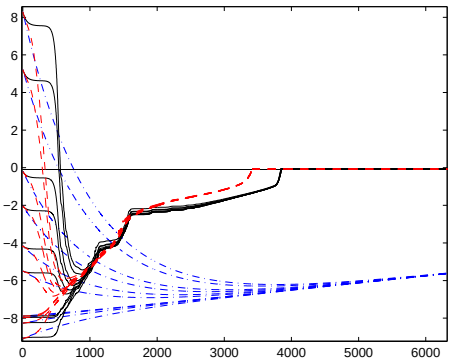

Fig. 18. Evoluzioni temporali degli stati $x_i(k), i = 1, ..., N$ per $N = 10$ e coefficienti come in 56 ottenute con ε costante pari a 0.001, di poco inferiore a ε^*, (in blu a tratti) e con ε variabile secondo l'Alg. 7 (in nero) e secondo l'Alg. 8 (in rosso a tratti). Le modifiche apportate in ques'ultima versione dell'algoritmo, hanno consentito un ulteriore miglioramento.

Algorithm 8 Algoritmo di tuning del parametro ε perfezionato

1: $\boldsymbol{\delta}(h-\tau) = \left|\frac{\mathbf{y}(h-\tau)}{\mathbf{z}(h-\tau)} - \frac{\mathbf{y}(h-\tau-\bar{\tau})}{\mathbf{z}(h-\tau-\bar{\tau})}\right|$
2: $\boldsymbol{\delta}(h) = \left|\frac{\mathbf{y}(h)}{\mathbf{z}(h)} - \frac{\mathbf{y}(h-\tau)}{\mathbf{z}(h-\tau)}\right|$
3: **if** $\sum_{i=1}^{N} \delta_i(h-\tau) > \sum_{i=1}^{N} \delta_i(h)$ **then**
4: **if** ε è stato aumentato precedentemente **then**
5: $\bar{\varepsilon} = \varepsilon$ ▷ continua ad aumentarlo
6: **if** $\varepsilon < 0.5$ **then**
7: $\varepsilon = 1.5\varepsilon$
8: **else**
9: $\varepsilon = \varepsilon + (1-\varepsilon)/2.5$
10: **end if**
11: **else** ▷ se in precedenza è diminuito,
12: $\Delta_\varepsilon = |\bar{\varepsilon} - \varepsilon|$
13: $\bar{\varepsilon} = \varepsilon$
14: $\varepsilon = \varepsilon + \Delta_\varepsilon/2$ ▷ riportalo a un valore intermedio
15: **end if**
16: $\tau = \tau/2$
17: **if** $\tau < 1$ **then**
18: $\tau = 1$
19: **end if**
20: **else**
21: **if** ε è stato aumentato precedentemente **then**
22: $\Delta_\varepsilon = |\bar{\varepsilon} - \varepsilon|$
23: $\bar{\varepsilon} = \varepsilon$
24: $\varepsilon = \varepsilon + \Delta_\varepsilon/2$ ▷ riducilo ad un valore intermedio
25: **else** ▷ se in precedenza è diminuito,
26: $\varepsilon = \varepsilon/2$ ▷ continuna a diminuirlo
27: $\bar{\varepsilon} = \varepsilon$
28: **end if**
29: $\tau = \tau * 2$
30: **end if**

L'idea che è stata seguita è quella di utilizzare la stessa stategia di tuning precedentemente illustrata in parallelo per ogni agente, con l'unica differenza che il valore di ε_i viene aggiornato in base allo stato x_i del singolo agente. Il codice con i dovuti aggiustamenti è presentato in Alg. 9[18].

Ora ogni agente procede con il tuning del proprio parametro ε_i in modo indipendente e sfasato rispetto agli altri. Questo perché ogni agente esegue l'aggiornamento ogni τ_i iterazioni e aggiusta il valore di τ_i solo in base al movimento della propria variabile di stato. Non è necessario lo scambio di alcuna informazione aggiuntiva tra i diversi agenti e quindi non si ha perdita di efficienza. Tuttavia emergono altri tipi di problemi. Dato che gli agenti non comunicano sempre tra loro, come esposto nella Sez. II-D3, la convergenza al punto di minimo globale è ritardata. Inoltre la casualità con cui avvengono gli scambi di dati si traduce in evoluzioni di stato caratterizzate da andamenti disomogenei. Tutto ciò si ripercuote sul processo di tuning dei vari ε_i con effetti molto diversi per ogni singolo agente. Risulta possibile, infatti, che un nodo della rete entri in fase convergente prima degli altri e quindi alzi in modo eccessivo il proprio parametro rischiando di ricadere in fase divergente che causerebbe uno spreco di iterazioni. In alcuni casi, la procedura con cui vengono aggiornati i periodi τ_i può portare a un aumento spropositato degli stessi. L'effetto che ne deriva è quello di un peggioramento delle prestazioni dovuto sia alla lentezza nell'aggiornamento di ε_i, sia al fatto che un periodo di campionamento troppo lungo maschera, di fatto, la reale evoluzione dello stato.

La soluzione a questo problema non è, però, difficile da trovare. Infatti è sufficiente introdurre un valore massimo per

[18]Si noti che in questo caso nella riga di codice 3 non compaiono sommatorie. Questo perché ogni agente prende in considerazione esclusivamente l'evoluzione della propria variabile di stato. Ciò ha lo spiacevole effetto di aumentare la sensibilità alle rapide variazioni del singolo stato, ma è necessario per realizzare un algoritmo distribuito

Algorithm 9 Algoritmo distribuito di tuning del parametro ε

$\delta_i(h-\tau_i) = \left| \frac{yi(h-\tau_i)}{zi(h-\tau_i)} - \frac{yi(h-\tau_i-\bar{\tau}_i)}{zi(h-\tau_i-\bar{\tau}_i)} \right|$
$\delta_i(h) = \left| \frac{yi(h)}{zi(h)} - \frac{yi(h-\tau_i)}{zi(h-\tau_i)} \right|$
if $\delta_i(h-\tau) > \delta_i(h)$ **then**
    if ε_i è stato aumentato precedentemente **then**
        $\bar{\varepsilon}_i = \varepsilon_i$
        if $\varepsilon_i < 0.5$ **then**
            $\varepsilon_i = 1.5\varepsilon_i$
        else
            $\varepsilon_i = \varepsilon_i + (1-\varepsilon_i)/2.5$
        end if
    else
        $\Delta_{\varepsilon_i} = |\bar{\varepsilon}_i - \varepsilon_i|$
        $\bar{\varepsilon}_i = \varepsilon_i$
        $\varepsilon_i = \varepsilon_i + \Delta_{\varepsilon_i}/2$
    end if
    $\tau_i = \tau_i/2$
    if $\tau_i < 1$ **then**
        $\tau_i = 1$
    end if
else
    if ε è stato aumentato precedentemente **then**
        $\Delta_{\varepsilon_i} = |\bar{\varepsilon}_i - \varepsilon_i|$
        $\bar{\varepsilon}_i = \varepsilon_i$
        $\varepsilon_i = \varepsilon_i + \Delta_{\varepsilon_i}/2$
    else
        $\varepsilon_i = \varepsilon_i/2$
        $\bar{\varepsilon}_i = \varepsilon_i$
    end if
    $\tau_i = \tau_i * 2$
end if

i periodi τ. Esso può essere visto come un iperparametro dell'algoritmo e non necessita di essere individuato con grade precisione. Può essere introdotto anche nella versione dell'algoritmo con un unico ε comune, ma non ha un impatto altrettanto efficace sulle prestazioni (in termini di numero totale di iterazioni) come quello riscontrato in questo caso. La motivazione risiede proprio nel fatto che gli agenti operano ciascuno per conto proprio e si influenzano solamente grazie al *consensus*, cioè non è più presente quell'opera di bilanciamento rappresentata dall'avere un unico parametro in comune che presentava il vantaggio di moderare le variazioni più brusche. In compenso si è raggiunto il grande risultato di ottenere un algoritmo in grado di operare in maniera totalmente distribuita. Esso è esposto in Alg. 10.

Osservando in Fig. 19, l'evoluzione di stato ottenuta eseguendo l'Alg. 10 (in rosso) mostra andamenti abbastanza differenziati per ogni agente ed essi appaiono molto meno regolari rispetto agli altri casi. A differenza del caso centralizzato, le prestazioni non sono migliori di quelle raggiungibili

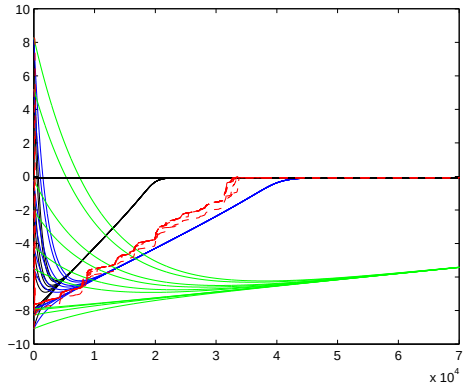

Fig. 19. Evoluzioni temporali degli stati $x_i(k), i = 1, ..., N$ per $N = 10$ e coefficienti come in 56 ottenute con ε costante pari a 0.001, di poco inferiore a ε^*, (in nero), con ε variabile secondo l'Alg. 10 con $\tau_{max} = 32$ (in rosso a tratti) e di nuovo con ε costante pari a 0.0005 (in blu) e con ε costante pari a 0.0001 (in verde). In questo caso l'evoluzione ottenuta con la procedura proposta, non è quella che presenta la più alta velocità di convergenza, ma essa risulta comunque soddisfacente e, cosa ben più importante, non è rilevante il valore iniziale del parametro da cui parte l'algoritmo.

tenendo costante il parametro ε per ogni agente impostandolo ad un valore leggermente inferiore a ε^*, ma sono comunque buone rispetto a quelle che si avrebbero settando ε a valori inferiori.

F. Conclusioni

Quello proposto è un criterio per il tuning del parametro ε applicabile anche alla versione distribuita dell'algoritmo presentato in 1 che consente di aggirare il problema dell'individuazione del valore critico ε^*. Indipendentemente, infatti, dal valore di partenza assegnato ad ε, si converge al punto di minimo cercato in un numero relativamente contenuto di iterazioni. ottenendo, quindi, prestazioni non troppo distanti dal miglior caso possibile per la versione a parametro costante.

REFERENCES

[1] F. Zanella, D. Varagnolo, A. Cenedese, G. Pillonetto, L. Schenato, "Newton-Raphson consensus for distributed convex optimization", *http://paduareserch.cab.unipd.it/*, Tech Rep., 2011.

[2] D. P. Bertsekas, A. Nedić, E. Ozdaglar, "Convex Analysis and Optimization", Athena Scientific, 2003.

[3] S. Boyd, A. Ghosh, B. Prabhakar, D. Shah, "Gossip Algorithms: Design, Analysis and Applications", Information Systems Laboratory, Stanford University.

[4] S. Boyd, N. Parikh, E. Chu, B. Peleato, 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.

[5] S. Boyd, L. Xiao, A. Mutapcic, J. Mattingley, "Notes on Decomposition Methods", Notes for EE364B, Stanford University, 2007.

[6] S. Boyd, L. Vandenberghe, "Convex Optimization", Cambridge University Press, 2004.

[7] R. Carli, F. Fagnani, P. Frasca, S. Zampieri, "Gossip consensus algorithms via quantized communication", *International Journal of Robust and Non-Linear Control*, 19(16):1787-1816, 2009.

[8] F. Fagnani, S. Zampieri, "Randomized consensus algorithms over large scale networks", *IEEE Journal of Selected Areas in Communications*, 26(4):634-649, 2008.

Algorithm 10 Versione finale dell'algoritmo distribuito di tuning del parametro ε

$\delta_i(h-\tau_i) = \left| \frac{yi(h-\tau_i)}{zi(h-\tau_i)} - \frac{yi(h-\tau_i-\bar{\tau_i})}{zi(h-\tau_i-\bar{\tau_i})} \right|$
$\delta_i(h) = \left| \frac{yi(h)}{zi(h)} - \frac{yi(h-\tau_i)}{zi(h-\tau_i)} \right|$
if $\delta_i(h-\tau) > \delta_i(h)$ **then**
    if ε_i è stato aumentato precedentemente **then**
        $\bar{\varepsilon_i} = \varepsilon_i$
        if $\varepsilon_i < 0.5$ **then**
            $\varepsilon_i = 1.5\varepsilon_i$
        else
            $\varepsilon_i = \varepsilon_i + (1-\varepsilon_i)/2.5$
        end if
    else
        $\Delta_{\varepsilon_i} = |\bar{\varepsilon_i} - \varepsilon_i|$
        $\bar{\varepsilon_i} = \varepsilon_i$
        $\varepsilon_i = \varepsilon_i + \Delta_{\varepsilon_i}/2$
    end if
    $\tau_i = \tau_i/2$
    if $\tau_i < 1$ **then**
        $\tau_i = 1$
    end if
else
    if ε è stato aumentato precedentemente **then**
        $\Delta_{\varepsilon_i} = |\bar{\varepsilon_i} - \varepsilon_i|$
        $\bar{\varepsilon_i} = \varepsilon_i$
        $\varepsilon_i = \varepsilon_i + \Delta_{\varepsilon_i}/2$
    else
        $\varepsilon_i = \varepsilon_i/2$
        $\bar{\varepsilon_i} = \varepsilon_i$
    end if
    $\tau_i = \tau_i * 2$
    if $\tau_i > \tau_{max}$ **then** ▷ non superare il periodo massimo
    else
        $\tau_i = \tau_{max}$
    end if
end if

[9] F. Garin, L. Schenato, "A survey on distributed estimation and control applications using linear consensus algoritmhs", Networked Control Systems, Springer, 2011.

[10] F. Girosi, "An Equivalence Between Sparse Approximation and Support Vector Machines", Department of Brain and Cognitive Sciences, Massachusetts Institute of Technology, 1997.

[11] S.R. Gunn, "Support Vector Machines for Classification and Regression", Technical Report, Faculty of Engineering and Applied Science, University of Southampton, 1998.

[12] B. Johansson - "On distributed optimization in networked systems", Ph.D. dissertation, KTH Elettrical Engineering, 2008.

[13] H. K. Khalil, "Nonlinear Systems (third edition)", Prentice Hall, 2002.

[14] J. Lin, E. Elhamifar, I-J. Wang, R. Vidal, "Consensus with Robustness to Outliers via Distributed Optimization", John Hopkins University, Baltimora, US.

[15] A. Nedić, D.P. Bertsekas, "Incremental subgradients methods for non-differentiable optimization", *SIAM J. on Optim.*, Vol.12, No.1, pp.109-138, 2001.

[16] G. Picci, "Metodi statistici per l'identificazione di sistemi lineari", dispensa delle lezioni, 2011.

[17] M. Rabbat, R. Novak, "Distributed Optimization on Sensor Networks", Department of Electrical and Computer Engineering, University of Winsconsin, 2004.

[18] S. S. Ram, A. Nedić, V. V. Veeravalli, "Incremental Stochastic Subgradient Algorithms for Convex Optimization", *SIAM J. on Optim.*, Vol.20, No.2, pp.691-717, 2009.

[19] W. Ren, R. Beard, "Distributed Consensus in Multi-vehicle Cooperative Control, Theory and Applications", Springer, 2008.