Passando dalla formalizzazione matematica vista nella sezione precedente giungiamo alla sua implementazione. In algorithm1 è riportato lo pseudocodice che da i dati di input e la loro previsione, calcola i valori dei moltiplicatori lagrangiani e successivamente permette di trovare il valore di W e b.  
L’obiettivo è quindi quello di rappresentare il problema di massimizzazione formalizzato in (6). Primo passaggio è trasformare la ricerca di un massimo nella ricerca di un minimo che si risolve cambiando il segno della nostra funzione. Successivamente vanno definite le matrici, e i constraint che ci permettono di rappresentare il QP:

[FORMULA] min 1/2\*x’Qx + qx lower\_bound <= x <= upper\_bound

Analizziamo quindi i singoli componenti come vengono formalizzati:

* **Q**: da (6) si nota che è rappresentato dalla matrice di Kernel. Dato che alcuni kernel generano una matrice che non risulta essere *positive semi-definite*, basti pensare al caso in cui il kernel scelto contiene *tanh*; la formalizzazione che viene usata all’interno dello pseudocodice ci assicura di ottenere sempre una matrice con questa caratteristica.  
  DUBBIO: Quindi la forma PSD di Q ci assicura qualche proprietà rispetto alla soluzione del QP? COSA???  
  Che quindi imo è la risposta a perché dobbiamo stare attenti e non possiamo usare solo K
* **q**: viene costruita in modo da unire tra loro entrambi gli elementi che contengo *alfa* e *alfa\**
* **linear constraint**: il vincolo lineare presente sui moltiplicatori lagrangiani ci permette di definire lb=0 e ub=C

Data questa formalizzazione tramite algorithm2 verrà risolto il problema di minimizzazione ottenendo i valori dei moltiplicatori lagrangiani. A questo punto sarà quindi possibile calcolare il valore di W e b in modo tale da rappresentare la nostra funzione

[FORMULA

] F(x) = wx +b = SUM{i] (−αi + αi∗)K(xi , x) + b

DOMANDA: è veramente necessario definire w? Che poi non dipende dal valore di x che consideriamo per calcolare l’output? Quindi non abbiamo una forma di w generale o no?

Per determinare il valore di b andremo ad applicare quanto già evidenziato precedentemente nella sezione 1 in (FORMULA DI B AGGIUNGI REF), sarà necessario trovare un elemento che si trovi al di fuori dell’e-tube e rispetto quest’ultimo calcolare il valore.  
DUBBIO: Abbiamo letto diversi paper in letteratura e abbiamo trovato diversi modi in cui viene determinato il valore b. Alcuni dicono che basta usare un punto che sia fuori dall’e-tube, altri invece lo calcolano su tutti i punti per poi fare una media. C’è un metodo più corretto?

DUBBIO BETA: Nel calcolo del Polyak stepsize utilizziamo un parametro *beta* che nel caso di deflection first deve essere minore uguale del coefficiente di deflection. Nelle slide e appunti però non abbiamo visto nessuna formula rispetto la quale è definito, esiste?

DUBBIO GAMMA: Abbiamo visto studiando dagli appunti presi a lezione e dalle slide che *gamma* viene definito nel seguente modo

[FORMULA] argmin { … }

Come lei ha detto non è obbligatorio avere un valore di gamma variabile e che si adatta a quello che è il comportamento della nostra ricerca del minimo, anche se avere un valore adattivo ci permette di ottenere risultati migliori. Dobbiamo quindi analizzare entrambe le possibilità? Nel caso in cui dobbiamo risolvere (FORMULA) andremo a calcolare per ogni iterazione nel nostro algoritmo di ricerca un problema di minimizzazione? Non diventa “pesante” dal punto di vista della computazione?

RIGUARDO L’ALGORITMO

Come richiesto dalla task utilizzeremo un \texit{*subgradient method},* il quale viene utilizzato per trattare funzioni che possono anche avere punti di \textit{nondifferenziabilità}. E’ dunque uno dei metodi più semplici e con meno richieste nei confronti di proprietà necessarie per f. In particolare si distingue dagli ordinari metodi di gradiente per 2 aspetti:

* lo \textit{stepsize} non viene scelto tramite \texit{line search} (PERCHE? Capisco che line search utilizza gradiente a buso quindi è probabilmente per questo, ma è solo per questo?) ma tramite altri ragionamenti (il più immediato tra tutti è \textbf{DSS}, dove lo stepsize viene fissato a priori con degli appropriati vincoli di scelta)
* non è un \textit{descent method}, ovvero il valore della funzione può incrementare nelle iterazioni. Questo avviene dato che non tutti i subgradient portano a direzioni di discesa. Anche se questo è vero abbiamo provato durante il corso che iterazioni successive ci portano ad avvicinarci a x\* monotonicamente. Con questa certezza sappiamo che convergeremo lentamente ma sicuramente ad x\* e di conseguenza anche ad f\* (anche se probabilmente non monotonicamente). [https://web.stanford.edu/class/ee364b/lectures/subgrad\_method\_notes.pdf]

Entrando più nel dettaglio per quanto riguarda lo \textit{stepsize}, la scelta di fissarlo a priori (DSS) ci porta ad una convergenza estremamente lenta, dunque preferiremmo utilizzare altre euristiche. In particolare, se avessimo a disposizione f\* potremmo seguire la \textit{*Polyak stepsize} (\textbf{PSS})*, la quale ci permette di determinare lo stepsize ottimo basandosi sulla seguente formula:

(inserisci formula)

Osserviamo a questo punto l’ulteriore parametro $\beta$, il quale deve rispettare il vincolo $\beta \in [0,2]$ e che per la massima efficienza di convergenza viene posto (utilizzando PSS base, ovvero non il nostro caso) a 1.

Purtroppo nel nostro caso non siamo a conoscenza di f\* e dobbiamo di conseguenza introdurre la tecnica che verrà utilizzata in Algorithm2, ovvero \textit{*Target Level Stepsize}*. Essa si basa sullo stimare f\* per poter poi attuare la stessa formulazione di Polyak, rimanendo sempre pronto a rivalutare la stima nel momento in cui una nuova e migliore stima viene ottenuta.

Più nel dettaglio la tecnica Target Value utilizza dei parametri \rho e \delta, tramite i quali, dopo aver fissato una stima iniziale \textit{fref}, ci permette di aggiornare la stima secondo i seguenti concetti:

* \delta (parametro che segue il vincolo \delta > 0) ci permette di definire una stima di f\* 🡪 fref - \delta tale per cui la stima fref verrà aggiornata solo se troveremo un valore f che migliori decisamente la stima (ovvero f < fref - \delta).
* \rho (parametro che segue il vincolo \rho \in (0,1)) ci permette di ottenere un vanishing \delta, ovvero dopo aver superato una certa threshold andremo a rimpicciolire \delta seguendo 🡪 \delta = \rho \* \delta. Questo viene effettuato perché più ci avvicineremo alla soluzione più il range che ci distanzia da f\* rimpicciolisce, lasciando meno spazio di miglioramento ad f.

L’altro elemento richiestoci dalla task è l’utilizzo di un \textit{deflected method}. Questa metodologia ci permette di basare il nuovo punto di valutazione di f (ovvero x\_i) non solo dando peso al subgradient calcolato ma anche basandosi sulla precedente direzione intrapresa, evitando un andamento esageratamente a zig zag. Per pesare le 2 parti necessitiamo di un parametro \gamma (vincolato tra 0 e 1, 0 🡪 solo deflected, 1 🡪 solo gradiente), arrivando alla formulazione:

(inserisci formula)

In particolare all’interno di Algorithm2 andremo ad utilizzare l’approccio deflection-first/stepsize-restricted, ovvero ad ogni iterazionedefiniremo innanzitutto la direzione d da intraprendere seguendo (formula sopra) per poi ottenere lo stepsize:

(inserisci formula)

Vincolando \beta 🡪 \beta < \gamma.