Laboratorio Didattico di Matematica Computazionale

a.a. 2018-2019

Esercitazione 10

Problemi di massimo e minimo

Per questa esercitazione è richiesto l'upload del codice di uno a scelta tra l'Esercizio 2 e l'Esercizio 4.

Sia $f : [a, b] \to \mathbb{R}$ una funzione continua (se necessario anche derivabile un opportuno numero di volte). Ci poniamo il problema di calcolare numericamente i punti di massimo e/o minimo di f.

Per esempio, f potrebbe rappresentare il costo di un progetto in funzione di un certo parametro; determinarne il minimo ci permette di scegliere il parametro in modo che il costo del progetto sia il più basso possibile.

Più in generale, possiamo considerare il caso in cui f è una funzione a valori reali definita su un opportuno dominio chiuso e limitato di $\mathbb{R}^n$, per esempio su un rettangolo in $\mathbb{R}^2$ o un parallelepipedo in $\mathbb{R}^3$

A volte è facile determinare analiticamente massimi e minimi, ma in altri casi f è eccessivamente complicata (o non se ne conosce neppure un'espressione esplicita) ed è quindi necessario affidarsi a metodi numerici.

Senza perdita di generalità, ci limiteremo a studiare problemi di minimo. I metodi numerici di minimizzazione (ottimizzazione) sono essenzialmente di due tipi: quelli che calcolano la derivata (o il gradiente) di f e quelli che invece usano solo valutazioni di f, senza calcolare derivate.

1. Discretizzazione

Un approccio molto semplice consiste nel discretizzare $[a, b]$ scegliendo dei punti equispaziati $x_0 = a < x_1 < \ldots < x_{N-1} < x_N = b$, quindi valutare f su tutti gli x_j e cercare il minimo o massimo. A volte questo metodo può dare buoni risultati, ma in altri casi rischia di essere dispendioso e poco efficace, perché potrebbe richiedere una discretizzazione molto fine dell'intervallo per dare risultati accurati, e conseguentemente molte valutazioni della f. Se poi f, anziché essere definita su un intervallo, fosse definita su un dominio 2 o 3-dimensionale, il costo computazionale crescerebbe ulteriormente.

Esercizio 1 *Come esercizio di riscaldamento, scrivete una function* `forzabruta.m` *che prenda in ingresso una handle* `fun` *a una funzione di una variabile, un intervallo di definizione* `[a,b]` *e l'intero positivo* `N`*, e restituisca in output il punto x_j in cui si ha il minimo, secondo l'algoritmo descritto sopra, e il valore corrispondente della funzione.*

Verificate il buon funzionamento del codice su una funzione "facile", per esempio $f(x) = x^2 + 1$ definita su $[-1, 1]$.

Provate poi ad applicare la function alla funzione $f(x) = \frac{1}{x} \sin\left(\frac{1}{x}\right)$ su $[0.08, 3]$ e confrontatene l'efficacia rispetto al comando `fminbnd` *di Matlab, che usa l'interpolazione parabolica. Quante valutazioni sono necessarie nei due casi per avere un'approssimazione ugualmente buona del minimo?*

2. Steepest descent

Ci proponiamo qui di implementare una semplice versione del metodo della steepest descent applicato ad una funzione differenziabile $f : \mathbb{R}^2 \to \mathbb{R}$. Si tratta di un metodo iterativo. L'idea è di "scendere a valle" lungo il grafico di f seguendo la direzione data da $-\nabla f$. Ciascun passo è quindi della forma

$$(x_{new}, y_{new}) = (x_{old}, y_{old}) - \alpha \nabla f(x_{old}, y_{old}),$$

dove la lunghezza del passo $\alpha > 0$ è scelta in modo opportuno.

Il programma riceve in input:

- `fun`: handle alla funzione da minimizzare,
- `grad`: handle al gradiente della funzione da minimizzare,
- `init`: vettore di lunghezza 2 contenente le coordinate del punto iniziale,
- `step`: numero reale positivo corrispondente alla lunghezza del passo iniziale,
- `[a,b], [c,d]`: estremi della "finestra" in cui si cerca il minimo (in alternativa, per una finestra circolare di centro `init`, si assegnerà il raggio `R`),
- `itermax`: numero massimo di iterazioni,
- `tol`: tolleranza sulla norma del gradiente, da usare come criterio di convergenza.

Come abbiamo detto, il metodo che vogliamo implementare è di tipo iterativo, quindi sarà necessario inserire un ciclo nel programma. Ad ogni iterazione, il programma ha a disposizione un'approssimazione attuale del minimo `xmin` e un valore `step` della lunghezza del passo, e fa quanto segue:

- calcola il gradiente in `xmin`,
- esegue un passo di lunghezza `step`, ottenendo un punto `x1`,
- valuta se il passo eseguito è accettabile: se sì, si pone `xmin=x1` e l'iterazione è terminata, invece se `x1` esce dalla finestra, oppure non ha diminuito la funzione, si dimezza la lunghezza del passo e si rifà l'iterazione.

L'iterazione si arresta se:

- la norma del gradiente è minore di `tol` (convergenza),
- oppure il numero di iterazioni eseguite è maggiore di `itermax` (fallimento).

Naturalmente è sempre bene stampare l'esito dell'esecuzione, in modo che l'utente sappia se il programma si è arrestato perché ritiene di aver trovato un minimo, o perché non riesce a risolvere il problema.

Provate poi a introdurre una modifica per vedere se possiamo usare un passo più lungo e accelerare così l'iterazione. Se il passo eseguito per calcolare `x1` è subito accettabile, si raddoppia il passo e si rifà il controllo: se anche il nuovo punto calcolato è accettabile lo si tiene come nuova approssimazione del minimo e si mantiene il passo raddoppiato.

Esercizio 2 *Scrivere una function* `steepest.m` *che implementi il metodo delineato sopra. Scrivere uno script che applichi la function alla funzione* $f(x,y) = x^2/2 + 2y^2$ *su* $[-2,2] \times [-2,2]$ *e crei due figure: una contenente il grafico di* f *(cioè una superficie in* $\mathbb{R}^3$*) e una in cui si disegnano le curve di livello di* f *insieme al percorso seguito dal metodo.*

Esercizio 3 *Un metodo più sofisticato è implementato in Matlab sotto il nome di* `fminunc`. *Usate l'help per capire come funziona questo comando e provate ad applicarlo alla funzione dell'esercizio e alla funzione di Rosenbrock, disegnando in entrambi i casi il percorso sulle curve di livello. Che cosa osservate?*

La funzione di Rosenbrock è definita da $f(x,y) = 100(y-x^2)^2 + (1-x)^2$ e ha minimo in $(1,1)$. Per disegnarne il grafico potete scegliere per esempio il dominio di definizione $[-2,2] \times [-1,3]$.

3. Metodo di Nelder-Mead

In questa sezione vogliamo studiare un metodo di minimizzazione che non usa il gradiente. Si parte con tre punti $x_1, x_2, x_3 \in \mathbb{R}^2$ non allineati. Ad ogni iterazione il metodo si comporta nel modo seguente (provate a fare dei disegni per capire graficamente il significato di ciascuna operazione):

- rinomina i punti x_1, x_2, x_3 in modo da avere $f(x_1) \le f(x_2) \le f(x_3)$,
- calcola il punto medio x_m di x_1 e x_2,
- calcola il punto $x_r = 2x_m - x_3$ *(riflessione)*,
- ora abbiamo 3 casi:
  - 1. $f(x_1) \le f(x_r) < f(x_2)$: si rimpiazza x_3 con x_r e l'iterazione finisce,
  - 2. $f(x_r) < f(x_1)$: si calcola $x_e = 2x_r - x_m$ *(espansione)*,
  - se $f(x_e) < f(x_r)$, allora si rimpiazza x_3 con x_e e l'iterazione finisce,
  - altrimenti si rimpiazza x_3 con x_r e l'iterazione finisce,
  - 3. $f(x_r) \ge f(x_2)$: si calcola $x_c = (x_m + x_r)/2$ *(contrazione)*,
  - se $f(x_c) < f(x_3)$, allora si rimpiazza x_3 con x_c e l'iterazione finisce,
  - altrimenti si rimpiazza x_2 con $(x_2 + x_1)/2$ e x_3 con $(x_3 + x_1)/2$, e l'iterazione finisce.

Il programma termina quando:

- i punti x_1, x_2, x_3 sono sufficientemente vicini tra loro (convergenza),
- oppure se i valori di f su x_1, x_2, x_3 sono sufficientemente vicini tra loro (convergenza),
- oppure se il numero di iterazioni eseguite supera un valore prestabilito (fallimento).

Esercizio 4 *Scrivere una function* `neldermead.m` *che implementi il metodo delineato sopra. Scrivere uno script che applichi la function alla funzione di Rosenbrock e disegni le curve di livello della funzione insieme al percorso seguito dal metodo.*