Ricerca di minimi e massimi di funzioni

Rights & Credits

1 Introduzione

In qesta lezione ci proponiamo di studiare alcuni metodi per la ricerca di punti di minimo (min) e massimo (min) di funzioni, complessivamente chiamati punti di estremo.
Inizieremo col vedere i criteri per identificare min e max a partire dai valori delle derivate; poi accenneremo a qualche metodo per la ricerca di min e max, vedendo la semplice implementazione con Python di uno di essi; e infine useremo funzioni di SciPy per effettuare la ricerca di punti di estremo locale e globale di una funzione.

2 Identificazione di min e max con derivate

2.1 Definizione di min e max

Si dice che una funzione a valori reali f con dominio D, cioè $f : D \to \mathbb{R}$, ha un min globale, o min assoluto, in un punto $x_0 \in D$ se

$$f(x_0) \leq f(x) \qquad \forall x \in D \tag{1}$$

Analogamente, si dice che x_0 è un punto di max globale o max assoluto per $f(x)$ se

$$f(x_0) \geq f(x) \qquad \forall x \in D \tag{2}$$

Si noti che i min e max globali possono anche verificarsi agli estremi del dominio della funzione. Per esempio, è così per il max globale della funzione rappresentata in basso a destra nel dominio considerato.
Diciamo che x_0 è un punto di accumulazione in D se in ogni intorno di x_0 esiste almeno un altro elemento (punto) $x \in D$ diverso da x_0. In pratica, per la maggior parte dei casi e in particolare per quello rappresentato sotto, ciò significa che la funzione è definita con continuità nei dintorni di x.
Un punto di accumulazione x_0 è un punto di min locale per la funzione se vale la (1) in un intorno di x_0, cioè, in termini matematici, se esiste un $\epsilon > 0$ tale che $\forall x$ con $|x - x_0| < \epsilon$ valga la (1). Una definizione dello stesso tipo vale per il max locale con riferimento alla (2).

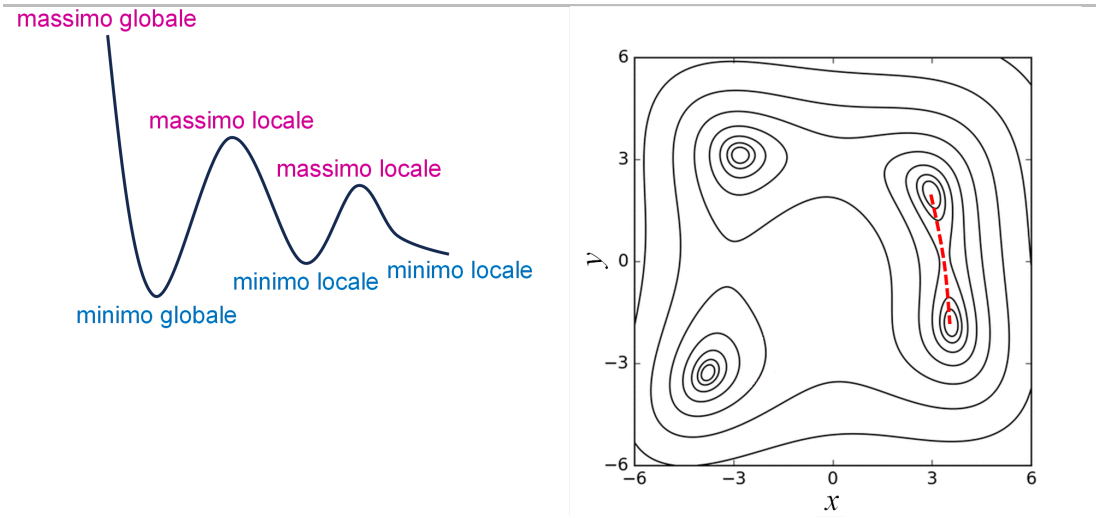


Figura a destra adattata da https://commons.wikimedia.org/wiki/File:Nelder-Mead_Himmelblau.gif.

2.1 Uso delle derivate per trovare min e max

Considerata una funzione sufficientemente regolare nel dominio di interesse (in tutta la trattazione seguente, la funzione sarà considerata continua almeno fino alle derivate del secondo ordine, cioè sarà una funzione almeno di classe C2), un punto di min o max locale x_0 interno al dominio è anche un punto stazionario, cioè un punto in cui

$$\left.\frac{df(x)}{dx}\right|_{x=x_0} = 0. \tag{3}$$

e quindi, localmente, la rapidità di variazione della funzione si riduce a zero. Ciò si vede facilmente dalla figura sopra a sinistra considerando il significato geometrico della derivata, cioè la tangente al grafico nel punto considerato. Chiaramente, non tutti i punti stazionari sono punti di min o max. Essi includono anche i punti di flesso orizzontale, in cui la derivata si annulla e poi continua a variare con lo stesso segno che aveva prima. Questo è, per esempio, il caso di quantità che caratterizzano alcune reazioni chimiche composite che avvengono a step successivi ben distinguibili sulla scala termporale. In conclusione, la stazionarietà espressa dall'equazione (3) è una condizione necessaria ma non sufficiente per stabilire l'esistenza di un punto di min o max per la funzione. Dobbiamo quindi considerare le derivate successive per asserire la natura del punto di stazionarietà.

Dalla figura vediamo che la pendenza della curva, quindi $f'(x)$, è negativa ma va aumentando (in quanto il suo modulo va diminuendo) prima del punto di min e continua ad aumentare dopo il punto di min (essendo positiva). Il fatto che la funzione derivata, $f'(x)$, aumenta nei dintorni del punto di min significa che la derivata di tale funzione, cioè $f''(x)$, è positiva. Concludiamo quindi che

$$\begin{cases} \left.\dfrac{df(x)}{dx}\right|_{x=x_0} = 0 \\ \left.\dfrac{d^2f(x)}{dx^2}\right|_{x=x_0} > 0 \end{cases} \quad \rightarrow \quad x_0 \text{ è un punto di min} \tag{4}$$

In modo analogo si vede che

$$\begin{cases} \left.\dfrac{df(x)}{dx}\right|_{x=x_0} = 0 \\ \left.\dfrac{d^2f(x)}{dx^2}\right|_{x=x_0} < 0 \end{cases} \quad \rightarrow \quad x_0 \text{ è un punto di max} \tag{5}$$

Il comportamento della funzione va esplorato ulteiormente se la sua derivata seconda si annulla pure a x_0. Ciò può essere anche fatto, semplicemente, considerando i valori della derivata prima negli intorni destro e sinistro del punto, ripercorrendo quindi il ragionamento fatto per giungere alle (4) e (5) in termini della derivata prima stessa (test della derivata prima).

Nel caso di una funzione di due variabili (che useremo come l'esempio più semplice di funzione di più variabili in questa analisi), si può descrivere la funzione geometricamente come una superficie. I suoi valori sono rappresentati lungo l'asse z, cioè $z = f(x, y)$, e il suo domonio costituisce, in genere, un sottoinsieme del piano x, y. In questo caso si possono anche avere punti di sella, in cui la funzione ha un minimo in una direzione e un massimo in una direzione ortogonale. Un esempio è mostrato sotto e un altro è rappresentato dal punto circa al centro della curva tratteggiata rossa nel grafico di contorno (*contour plot*) mostrato in alto a destra.

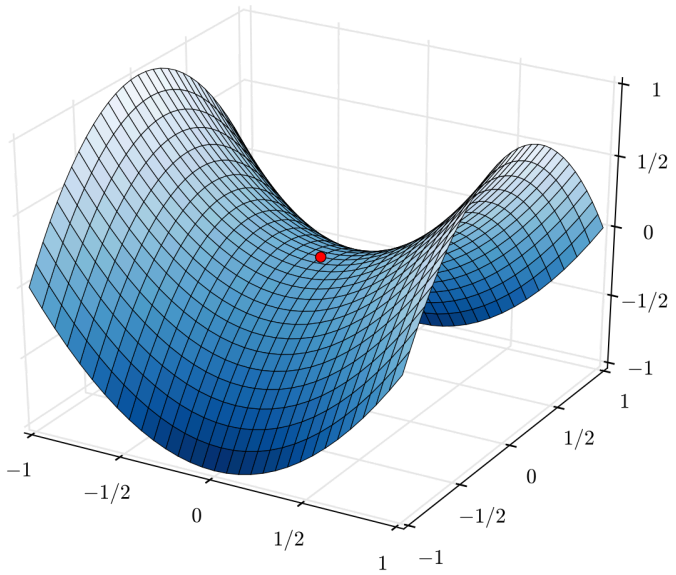


Immagine tratta da https://it.wikipedia.org/wiki/Punto_di_sella#/media/File:Saddle_point.svg.

Per caratterizzare i punti di min e max abbiamo adesso bisogno di più derivate parziali e dobbiamo riconsiderare la serie di Taylor troncata al secondo ordine mostrata

nell'equazione (17) della lezione precedente. A tal fine, notiamo che i termini quadratici si possono scrivere come il seguente prodotto matriciale:

$$\begin{aligned}
&\frac{1}{2}\begin{pmatrix}\Delta x & \Delta y\end{pmatrix}\begin{pmatrix}\dfrac{\partial^2 f}{\partial x^2} & \dfrac{\partial^2 f}{\partial x \partial y} \\ \dfrac{\partial^2 f}{\partial y \partial x} & \dfrac{\partial^2 f}{\partial y^2}\end{pmatrix}\begin{pmatrix}\Delta x \\ \Delta y\end{pmatrix} \\
&= \frac{1}{2}\begin{pmatrix}\Delta x & \Delta y\end{pmatrix}\begin{pmatrix}\Delta x\dfrac{\partial^2 f}{\partial x^2} + \Delta y\dfrac{\partial^2 f}{\partial x \partial y} \\ \Delta x\dfrac{\partial^2 f}{\partial y \partial x} + \Delta y\dfrac{\partial^2 f}{\partial y^2}\end{pmatrix} \\
&= \frac{1}{2}\left(\Delta x^2\frac{\partial^2 f}{\partial x^2} + 2\Delta x\Delta y\frac{\partial^2 f}{\partial x \partial y} + \Delta y^2\frac{\partial^2 f}{\partial y^2}\right)
\end{aligned} \tag{6}$$

con le derivate parziali calcolate nel punto di espansione. La matrice 2×2 che compare nella (6) si chiama matrice Hessiana:

$$\mathbf{H} = \begin{pmatrix}\dfrac{\partial^2 f}{\partial x^2} & \dfrac{\partial^2 f}{\partial x \partial y} \\ \dfrac{\partial^2 f}{\partial y \partial x} & \dfrac{\partial^2 f}{\partial y^2}\end{pmatrix} \tag{7}$$

Dato il vettore posizione attorno al quale si effettua lo sviluppo in serie di Taylor nella forma di vettore colonna

$$\mathbf{r}_0 = \begin{pmatrix}x_0 \\ y_0\end{pmatrix} \tag{8}$$

il vettore spostamento

$$\Delta\mathbf{r} = \begin{pmatrix}\Delta x \\ \Delta y\end{pmatrix} \tag{9}$$

e il gradiente nello spazio bidimensionale nella forma di vettore riga

$$\nabla f = \begin{pmatrix}\dfrac{\partial f}{\partial x} & \dfrac{\partial f}{\partial y}\end{pmatrix} \tag{10}$$

lo sviluppo di Taylor al secondo ordine si può scrivere nella forma compatta (trascurando la notazione di uguaglianza approssimata per uno spostamento sufficientemente piccolo)

$$f(\mathbf{r}_0 + \Delta\mathbf{r}) = f(\mathbf{r}_0) + \nabla f(\mathbf{r}_0)\cdot\Delta\mathbf{r} + \frac{1}{2}\Delta\mathbf{r}^T\mathbf{H}(\mathbf{r}_0)\Delta\mathbf{r} \tag{11}$$

che useremo per la seguente discussione. In tale equazione $\Delta\mathbf{r}^T$ rappresenta il vettore trasposto di $\Delta\mathbf{r}$, cioè un vettore riga. L'espressione $\Delta\mathbf{r}^T\mathbf{H}(\mathbf{r}_0)\Delta\mathbf{r}$ si chiama una forma quadratica. Considerando la (11) per uno spostamento sufficientemente piccolo, ci si può

fermare al primo ordine dello sviluppo e si può dimostrare che una condizione necessaria perché $\mathbf{r}_0$ sia un punto di min o max locale è

$$\nabla f(\mathbf{r}_0) = 0 \tag{12}$$

che è l'estensione della condizione (3) al caso bidimensionale. Quando questa condizione è soddisfatta, la (11) diventa

$$f(\mathbf{r}_0 + \Delta\mathbf{r}) = f(\mathbf{r}_0) + \frac{1}{2}\Delta\mathbf{r}^T\mathbf{H}(\mathbf{r}_0)\Delta\mathbf{r} \tag{13}$$

da cui è chiaro che, una volta soddisfatta la condizione (12):

- f ha un min locale in $\mathbf{r}_0$ se la forma quadratica $\Delta\mathbf{r}^T\mathbf{H}(\mathbf{r}_0)\Delta\mathbf{r}$ è definita positiva, cioè è maggiore di zero per qualsiasi spostamento $\Delta\mathbf{r}$; si dimostra che ciò accade se e solo se tutti gli autovalori della matrice $\mathbf{H}(\mathbf{r}_0)$ sono positivi. Studierete la nozione di autovalore nella parte di algebra lineare e poi potrete comprendere meglio l'affermazione precedente.
- f ha un max locale in $\mathbf{r}_0$ se la forma quadratica $\Delta\mathbf{r}^T\mathbf{H}(\mathbf{r}_0)\Delta\mathbf{r}$ è definita negativa, cioè è minore di zero per qualsiasi spostamento $\Delta\mathbf{r}$, il che è vero se e solo se tutti gli autovalori di $\mathbf{H}(\mathbf{r}_0)$ sono negativi.
- f ha un punto di sella in $\mathbf{r}_0$ se gli autovalori di $\mathbf{H}(\mathbf{r}_0)$ hanno segni diversi.
- Non si conosce la natura del punto se gli autovalori di $\mathbf{H}(\mathbf{r}_0)$ sono tutti nulli.

In realtà, le condizioni di sopra valgono anche in spazi a più dimensioni. Nel caso bidimensionale trattato qui, chiamato $\Delta\mathbf{H}(\mathbf{r}_0)$ il determinante della matrice Hessiana, tali condizioni si traducono nelle seguenti forme che, di per sé, non richiedono la conoscenza del concetto di autovalore.

Se è soddisfatta la condizione $\nabla f(\mathbf{r}_0) = 0$, allora:

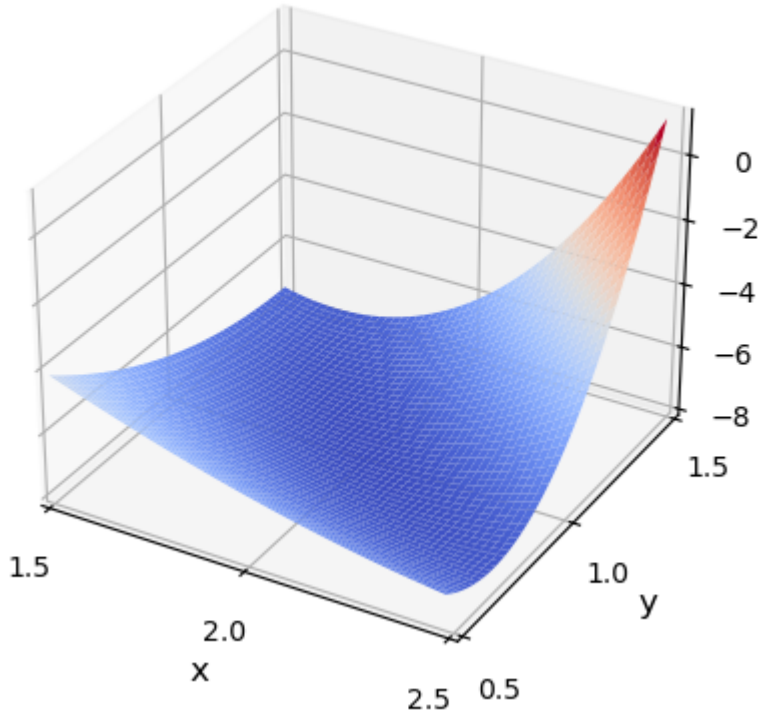
1. f ha un min locale in $\mathbf{r}_0$ se il primo elemento della matrice Hessiana calcolata in $\mathbf{r}_0$ è positivo e $\Delta\mathbf{H}(\mathbf{r}_0) > 0$,
2. f ha un max locale in $\mathbf{r}_0$ se il primo elemento della matrice Hessiana calcolata in $\mathbf{r}_0$ è negativo e $\Delta\mathbf{H}(\mathbf{r}_0) > 0$
3. f ha un punto di sella in $\mathbf{r}_0$ se $\Delta\mathbf{H}(\mathbf{r}_0) < 0$
4. ulteriore analisi è necessaria se $\Delta\mathbf{H}(\mathbf{r}_0) = 0$.

2.1.1 Esempio

Si consideri la funzione $f(x, y) = x^3y^2 - 12xy + 8y$

In [64]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
x = np.arange(1.5, 2.5, 0.02)
y = np.arange(0.5, 1.5, 0.02)
x, y = np.meshgrid(x, y)
```

```
z = x**3*y**2 - 12*x*y + 8*y
ax.plot_surface(x, y, z, cmap=cm.coolwarm)
ax.set_xlim(1.5,2.5)
ax.set_xticks(np.arange(1.5, 2.51, 0.5))
ax.set_xlabel('x', fontsize=12)
ax.set_ylim(0.5,1.5)
ax.set_yticks(np.arange(0.5, 1.51, 0.5))
ax.set_ylabel('y', fontsize=12);
```



Uguagliamo a zero le derivate parziali prime:

$$\begin{cases} \dfrac{\partial f}{\partial x} = 3x^2y^2 - 12y = 0 \\ \dfrac{\partial f}{\partial y} = 2x^3y - 12x + 8 = 0 \end{cases}$$

Dalla prima equazione otteniamo $x^2y = 4$ e, sostituendo nella seconda, $8x - 12x + 8 = 0$ da cui $x = 2$ e quindi $y = 4/x^2 = 1$. Pertanto, $(x_0, y_0) = (2, 1)$.

$$\frac{\partial^2 f}{\partial x^2} = 6xy^2, \quad \frac{\partial^2 f}{\partial x \partial y} = 6x^2y - 12, \quad \frac{\partial^2 f}{\partial y^2} = 2x^3$$

Di conseguenza,

$$\mathbf{H}(2,1) = \begin{pmatrix} 12 & 12 \\ 12 & 16 \end{pmatrix}$$

Essendo $\Delta\mathbf{H}(2,1) = 48$, la condizione 1 di sopra è soddisfatte e quindi $(2,1)$ è un punto di minimo per la funzione data.

3 Metodi per la ricerca di min

In questa sezione vedremo qualche metodo di ottimizzazione per trovare minimi di una funzione di una o più variabili.

3.1 Metodo del gradiente

Nell'equazione (11), per uno spostamento sufficientemente piccolo, possiamo trascurare il termine quadratico nello spostamento e scrivere, con buona approssimazione,

$$f(\mathbf{r}_0 + \Delta\mathbf{r}) - f(\mathbf{r}_0) = \nabla f(\mathbf{r}_0) \cdot \Delta\mathbf{r} = |\nabla f(\mathbf{r}_0)||\Delta\mathbf{r}| \cos\theta \qquad (14)$$

dove θ è l'angolo tra il gradiente e il vettore spostamento. Fissata la lunghezza (il modulo) dello spostamento, il prodotto al secondo membro ha il valore più negativo quando $\Delta\mathbf{r}$ è in direzione opposta al gradiente, per cui $\theta = \pi$ e $\cos\theta = -1$. Ora, al primo membro dell'equazione (14) abbiamo la variazione nel valore della funzione in conseguenza di tale spostamento. Si ha, quindi, la massima diminuzione della funzione spostandosi in direzione opposta al gradiente.
Stabiliamo la seguente procedura per realizzare il metodo del gradiente o metodo di discesa del gradiente (nome originario in inglese: *gradient descent*):

1. Supponiamo di partire da un punto $\mathbf{r}$ nel dominio della funzione, in cui la funzione assume un certo valore $f(\mathbf{r})$. In un calcolo col computer, tale punto potrà assumere valori discreti e sarà quindi caratterizzato da un indice. Essendo il vettore posizione iniziale, lo chiamiamo $\mathbf{s}_0$. Calcoliamo il gradiente della funzione in tale punto. A tal fine, possiamo usare una delle procedure viste nella lezione precedente per calcolare le derivate parziali che intervengono nel gradiente e quindi il gradiente al punto $\mathbf{s}_0$, che denotiamo con $\mathbf{d}_0$: $\mathbf{d}_0 = -\nabla f(\mathbf{s}_0)$.
2. Facciamo un primo step di dimensione a_0 in direzione opposta al gradiente, per cui il vettore spostamento (cioè la variazione del vettore posizione) sarà $\Delta\mathbf{s}_0 = -a_n\mathbf{d}_0$. Aggiorniamo quindi la posizione a $\mathbf{s}_1 = \mathbf{s}_0 - a_n\mathbf{d}_0$.
3. Reiteriamo lo step 2 finché non si annulla il gradiente; pertanto, in un ciclo del programma avremo istruzioni del tipo

$$\mathbf{s}_{k+1} = \mathbf{s}_k - a_k\mathbf{d}_k$$
$$k = k + 1$$

Procedendo in tal modo otteniamo una sequenze monotona decrescente $f_0 = f(\mathbf{s}_0) \geq f_1 \geq f_2 \geq \ldots$ di valori della funzione che desideriamo converga a un min locale. Se una funzione è convessa (un esempio molto semplice è quello di un paraboloide), allora il min trovato è un min globale. Il metodo della discesa del gradiente si può applicare a spazi con un qualsiasi numero di dimensioni. Tuttavia, talvolta esso può richiedere molte iterazioni per calcolare un min locale con la richiesta accuratezza. Approcci basati sul metodo di Newton e sul metodo del gradiente coniugato convergono in poche iterazioni, anche se il costo di ciascuna iterazione è maggiore (uno di tali approcci è il metodo BFGS usato tramite una funzione di SciPy in una lezione precedente per trovare il minimo di una funzione). Il metodo del gradiente coniugato è

una variante del metodo del gradiente che consente una convergenza più rapida grazie a una scelta diversa della direzione di discesa.

Breve nota aggiuntiva (opzionale): La convergenza a un min è garantita quando la funzione ha opportune proprietà (per esempio, quando la funzione è convessa e soddisfa la cosiddetta condizione di Lipschitz, che limita la variazione di una funzione) e si scelgono opportunamente i valori di a_i (per esempio, la sequenza di valori di a_i nel metodo di Barzilai-Borwein o una sequenza che soddisfi le cosiddette condizioni di Wolfe). Si noti pure che il metodo della discesa del gradiente può essere visto come un'applicazione del metodo di Eulero per la risoluzione dell'equazione differenziale ordinaria $x'(t) = -\nabla f(t, x(t))$.

3.2 Metodo Di Newton

Il metodo di Newton è stato originariamente concepito per il problema generale di trovare le radici di un'equazione

$$f(x) = 0 \tag{15}$$

dove $f(x)$ può essere una funzione reale o complessa e deve essere derivabile. Ovviamente, si parte dalla non conoscenza della/e soluzione/i di tale equazione, quindi da un punto x_0 in cui $f(x_0) \neq 0$. Lo sviluppo di Taylor al primo ordine di tale funzione dà

$$f(x) \equiv f(x_0 + \Delta x) = f(x_0) + f'(x_0)\Delta x \equiv f(x_0) + f'(x_0)(x - x_0) \tag{16}$$

Se lo spostamento Δx ci portasse direttamente al punto giusto, cioè a una soluzione dell'equazione (15), allora sarebbe $f(x) = 0$. Possiamo considerare, quindi, la soluzione dell'equazione

$$f(x_0) + f'(x_0)(x - x_0) = 0 \tag{17}$$

cioè

$$x = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{18}$$

come un valore aggiornato migliore di x_0 per pervenire alla soluzione cercata. In termini di programma, con una delle consuete notazioni abbreviate $f_k^{(n)} = f^{(n)}[k] = f^{(n)}(x_k)$ per la funzione e le sue derivate calcolate in posizioni discrete (o, per esempio, a tempi discreti se si tratta di una funzione del tempo), dobbiamo effettuare l'iterazione

$$x_{k+1} = x_k - \frac{f_k}{f'_k}, \qquad k = 0, 1, 2, \ldots \tag{19}$$

La (19) è meglio nota come l'iterazione di Newton-Raphson ed è stata estesa da Simpson a una funzione vettoriale di più variabili della forma

$$\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_n(\mathbf{x})) = (0, 0, \ldots, 0) \equiv \mathbf{0} \tag{20}$$

dove $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. In tal caso la (19) diventa

$$\mathbf{x}_{k+1} = \mathbf{x}_k - J_k^{-1}\mathbf{f}_k, \qquad k = 0, 1, 2, \dots \tag{21}$$

dove $J_k \equiv J(\mathbf{x}_k)$ è la matrice Jacobiana

$$J(x_1, x_2, \dots, x_n) = \begin{pmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_n}{\partial x_1} & \cdots & \dfrac{\partial f_n}{\partial x_n} \end{pmatrix} \tag{22}$$

calcolata nel punto $\mathbf{x}_k$.

A questo punto, torniamo al problema di ricerca dei minimi di $f(x)$ sul suo dominio $D \in \mathbb{R}$. A tal fine, possiamo riformulare il problema di sopra per la derivata $g(x) = f'(x)$, che è anch'essa una funzione. Possiamo riscrivere quanto sopra per $g(x) = f'(x)$ e riottenere le equazioni (18) e (19) con g al posto di f, quindi sostituire g con f' per ottenere

$$x = x_0 - \frac{f'(x_0)}{f''(x_0)} \tag{23}$$

e la sua versione discreta allo step 0 e agli step successivi

$$x_{k+1} = x_k - \frac{f'_k}{f''_k}, \qquad k = 0, 1, 2, \dots \tag{24}$$

Alternativamente, si arriva allo stesso risultato procedendo come segue. Si considera lo sviluppo al secondo ordine della funzione.

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2 \tag{25}$$

x deve essere scelta in modo tale da minimizzare la funzione. Se la derivata seconda è positiva (per cui tale approssimazione quadratica rappresenta una funzione convessa nei dintorni di x_0), trovare il punto di min di f implica imporre che la funzione, espressa nella forma quadratica approssimata (25), abbia derivata nulla. In altre parole, prendendo la derivata di ambo i lati della (25), imponendo $f'(x) = 0$ e considerando che $f'(x_0)$ è una costante (essendo una funzione valutata in un punto) e quindi la sua derivata rispetto a x è nulla, risulta

$$0 = f'(x_0) + f''(x_0)(x - x_0) \tag{26}$$

da cui vien fuori di nuovo la (23).

Nel caso bidimensionale, bisogna applicare il gradiente all'equazione (11), che in realtà conserva la stessa forma quale che sia il numero di dimensioni. Dal momento che $\nabla f(\mathbf{r}_0)$ non dipende da $\mathbf{r}$, si ha

$$\begin{aligned}\nabla[\nabla f(\mathbf{r}_0)\cdot\Delta\mathbf{r}] &= \left(\hat{\mathbf{x}}\frac{\partial}{\partial x}+\hat{\mathbf{y}}\frac{\partial}{\partial y}\right)\left[\left.\frac{\partial f}{\partial x}\right|_{\mathbf{r}=\mathbf{r}_0}(x-x_0)+\left.\frac{\partial f}{\partial y}\right|_{\mathbf{r}=\mathbf{r}_0}(y-y_0)\right] \\ &= \hat{\mathbf{x}}\left.\frac{\partial f}{\partial x}\right|_{\mathbf{r}=\mathbf{r}_0}+\hat{\mathbf{y}}\left.\frac{\partial f}{\partial y}\right|_{\mathbf{r}=\mathbf{r}_0}=\nabla f(\mathbf{r}_0)\end{aligned} \tag{27}$$

La forma quadratica nella (11) ha l'espressione data nell'equazione (6), con le derivate calcolate nel punto $\mathbf{r}_0$. Per semplificare la notazione, scriviamo $H_{x_1x_2}=\left.\frac{\partial^2 f}{\partial x_1\partial x_2}\right|_{\mathbf{r}=\mathbf{r}_0}$ dove (x_1,x_2) può essere (x,x), (x,y), (y,x) o (y,y). Possiamo dunque scrivere

$$\begin{aligned}\nabla\left[\frac{1}{2}\Delta\mathbf{r}^T\mathbf{H}(\mathbf{r}_0)\Delta\mathbf{r}\right] &= \nabla\left(\frac{1}{2}H_{xx}\Delta x^2+H_{xy}\Delta x\Delta y+\frac{1}{2}H_{yy}\Delta y^2\right) \\ &= (H_{xx}\Delta x+H_{xy}\Delta y)\hat{\mathbf{x}}+(H_{yx}\Delta x+H_{yy}\Delta y)\hat{\mathbf{y}}\end{aligned} \tag{28}$$

ovvero, esprimendo il vettore al secondo membro come un vettore colonna,

$$\nabla\left[\frac{1}{2}\Delta\mathbf{r}^T\mathbf{H}(\mathbf{r}_0)\Delta\mathbf{r}\right]=\begin{pmatrix}H_{xx} & H_{xy}\\ H_{yx} & H_{yy}\end{pmatrix}\begin{pmatrix}\Delta x\\ \Delta y\end{pmatrix}=\mathbf{H}(\mathbf{r}_0)\Delta\mathbf{r} \tag{29}$$

In definitiva, l'applicazione del gradiente alla (11) dà

$$0=\nabla f(\mathbf{r}_0)+\mathbf{H}(\mathbf{r}_0)\Delta\mathbf{r} \tag{30}$$

dove il termine col gradiente è scritto come vettore colonna. Notiamo che, esplicitando l'espressione del vettore spostamento e ridistribuendo i termini a primo e secondo membro, la (30) si può immediatamente riscrivere nella forma

$$\mathbf{H}(\mathbf{r}_0)\mathbf{r}=\mathbf{H}(\mathbf{r}_0)\mathbf{r}_0-\nabla f(\mathbf{r}_0) \tag{31}$$

La (31) rappresenta un sistema lineare, scritto in forma vettoriale compatta, della forma

$$\begin{aligned}&\mathbf{A}\mathbf{r}=\mathbf{b} && (1)\\ &\text{con}\quad \mathbf{A}=\mathbf{H}(\mathbf{r}_0),\quad \mathbf{b}=\mathbf{H}(\mathbf{r}_0)\mathbf{r}_0-\nabla f(\mathbf{r}_0) && (2)\end{aligned} \tag{32}$$

Moltiplicando la (31) a sinistra per l'inversa della matrice Hessiana, si ottiene

$$\mathbf{r}=\mathbf{r}_0-\mathbf{H}^{-1}(\mathbf{r}_0)\nabla f(\mathbf{r}_0) \tag{33}$$

Si può dimostrare facilmente che la relazione (33) vale quale che sia il numero di dimensioni in gioco. Confrontando la (33) con la (23), si vede che la derivata è stata sostituita, come ci si aspetta, da un gradiente e la derivata seconda dalla matrice Hessiana, che contiene tutte le derivate parziali del secondo ordine. Il metodo di Newton è molto rapido, anche se non è garantito che converga a un punto di min in tutti i casi. Come si vede dalla (33), il metodo richiede che la derivata seconda o, in generale, la matrice Hessiana, sia invertibile. La risoluzione del sistema lineare (31), e quindi l'inversione della matrice Hessiana può essere effetuata efficacemente con il metodo del gradiente coniugato.

L'analogo dell'equazione (24) per funzioni di più variabili è

$$\mathbf{r}_{k+1}=\mathbf{r}_k-\mathbf{H}^{-1}(\mathbf{r}_k)\nabla f(\mathbf{r}_k),\qquad k=0,1,2,\ldots \tag{34}$$

Uno svantaggio del metodo di Newton è la necessità di calcolare le derivate prima e seconda di f o la matrice Hessiana; comunque, ove si usi tale metodo, il calcolo delle derivate necessarie può essere effettuato in modo esatto o discreto.
Vi sono, chiaramente, altri approcci all'ottimizzazione di una funzione che non trattiamo in questa lezione. Un altro metodo sarà menzionato nella sezione seguente.

4 Ricerca di min: esempi

4.1 scipy.optimize.minimize_scalar

Prima di tutto, importiamo le funzioni di SciPy e SymPy che ci serviranno e definiamo la funzione d'interesse. Tale funzione è piuttosto complicata e la sua derivata è ancora più complicata (vedi sotto), per cui sarebbe impossibile risolvere analiticamente l'equazione $f'(x) = 0$ e poi procedere con le derivate del secondo ordine per trovare i punti di min.
Nel seguito la stessa funzione è scritta in una forma, f, appropriata per essere usata dalle funzioni di Scipy e in una forma, g, adeguata all'uso di SimPy.

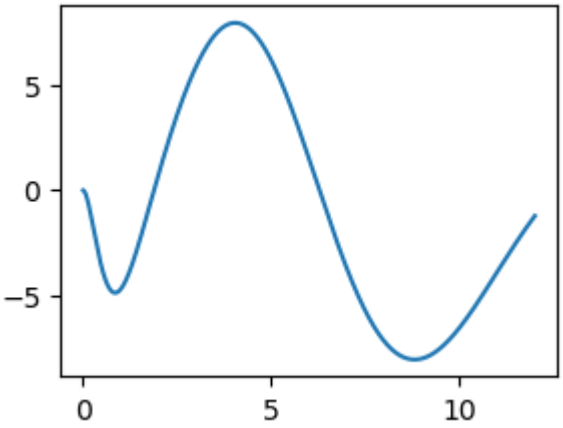
In [1]:
```
from scipy.optimize import minimize_scalar, basinhopping, minimize
from scipy.misc import derivative
import sympy as smp
from sympy import diff, symbols
x, y, f = symbols('x y f')

def f(x):
    return x**3*np.exp(-x/2)*np.sin(0.5*x)-40*x**2*np.exp(-2*x)*np.cos(0.5*x)-np

g = x**3*smp.exp(-x/2)*smp.sin(0.5*x)-40*x**2*smp.exp(-2*x)*smp.cos(0.5*x)-smp.a
g
```

Out[1]:
$$x^3 e^{-\frac{x}{2}} \sin(0.5x) - 40x^2 e^{-2x} \cos(0.5x) - \frac{\operatorname{atan}(x)}{x^2+3}$$

In [2]:
```
dg = diff(g,x)
ddg = diff(g,x,x)
dg
```

Out[2]:
$$-\frac{x^3 e^{-\frac{x}{2}} \sin(0.5x)}{2} + 0.5x^3 e^{-\frac{x}{2}} \cos(0.5x) + 20.0x^2 e^{-2x} \sin(0.5x) + 80x^2 e^{-2x} \cos(0.5x) + 3x^2 e^{-}$$

In [3]:
```
x1 = np.linspace(0,12,10**4)
y1 = f(x1)
plt.figure(figsize=(3.2,2.4))
plt.plot(x1,y1)
plt.show()
```



Usiamo adesso la funzione minimize_scalar del modulo scipy.optimize. Tale funzione consente di effettuare la minimizzazione locale di una funzione scalare di una variabile. Useremo la funzione con i valori assegnati agli argomenti secondo le impostazioni predefinite; pertanto l'unico input necessario sarà la funzione di cui trovare il min locale.

In [4]:
```
minimize_scalar(f)
```

Out[4]:
```
 message:
         Optimization terminated successfully;
         The returned value satisfies the termination criteria
         (using xtol = 1.48e-08 )
 success: True
     fun: -4.833586647031791
       x: 0.8496631041987057
    nit: 10
   nfev: 13
```

4.2 Metodo di Newton con derivate discrete e continue

Prima di tutto, usiamo il metodo di Newton calcolando le derivate prima e seconda con i metodi discreti più basilari:

In [5]:
```
def df(x,d):
    return (f(x+d)-f(x))/d

def ddf(x,d):
    return (f(x+d)-2*f(x)+f(x-d))/d**2

def min_f():
    x = float(input())
    d = float(input())
    tol = float(input())
    dx = 10*tol
    i = 0
    while dx > tol:
        x1 = x - df(x,d)/ddf(x,d)
        dx = abs(x1-x)
        x = x1
        i += 1
    m = print("Il punto di min è ", x, "il numero di step fatti è ", i)
    return m
```

In [6]:
```
min_f()
```

```
Il punto di min è  0.8496630540154352 il numero di step fatti è  6
```

Abbiamo usato la stessa soglia di tolleranza della funzione di Scipy. Si noti che il risultato è molto accurato ed è stato ottenuto in soli 6 step.
Ora calcoliamo la derivata con la funzione derivative del modulo scipy.misc.

In [7]:
```
def min_f_bis():
    x = float(input())
    d = float(input())
    tol = float(input())
    dx = 10*tol
    i = 0
    while dx > tol:
        x1 = x - derivative(f,x,d,n=1)/derivative(f,x,d,n=2)
        dx = abs(x1-x)
        x = x1
        i += 1
    m = print("Il punto di min è ", x, "il numero di step fatti è ", i)
    return m
```

In [8]:
```
min_f_bis()
```

```
Il punto di min è  0.8496631040005915 il numero di step fatti è  5
C:\Users\Agostino\AppData\Local\Temp\ipykernel_33204\449854612.py:8: DeprecationWarning: scipy.misc.derivative is deprecated in SciPy v1.10.0; and will be completely removed in SciPy v1.12.0. You may consider using findiff: https://github.com/maroba/findiff or numdifftools: https://github.com/pbrod/numdifftools
  x1 = x - derivative(f,x,d,n=1)/derivative(f,x,d,n=2)
```

Vediamo che l'accuratezza è ancora superiore a quella ottenuta con le derivate discrete grezze usate nella funzione di prima e che 5 step sono stati sufficienti per raggiungere l'obiettivo.
Infine, usiamo SymPy e quindi calcoliamo le derivate in modo esatto sfruttando le espressioni analitiche ottenute prima con tale modulo.

In [9]:
```
d_g = smp.lambdify(x,dg)
dd_g = smp.lambdify(x,ddg)

def min_f_tris():
    x = float(input())
    tol = float(input())
    dx = 10*tol
    i = 0
    while dx > tol:
        x1 = x - d_g(x)/dd_g(x)
        dx = abs(x1-x)
        x = x1
        i += 1
    m = print("Il punto di min è ", x, "il numero di step fatti è ", i)
    return m
```

In [10]:
```
min_f_tris()
```

```
Il punto di min è  0.8496631041277377 il numero di step fatti è  5
```

Come vedete, il valore è vicinissimo a quello ottenuto con la funzione `minimize_scalar` e 5 step sono stati di nuovo sufficienti.

4.3 scipy.optimize.basinhopping

La funzione basinhopping del modulo scipy.optimize consente di ottenere il minimo globale di una funzione usando l'algoritmo del salto di bacino (il nome inglese originale è *basin-hopping algorithm*). Esso è anche chiamato un metodo. L'algoritmo, concepito per mimare il processo naturale di minimizzazione dell'energia di raggruppamenti atomici, è considerato uno degli algoritmi più attendibili (e da taluni anche il più attendibile), in chimica fisica, per la ricerca della struttura di più bassa energia di raggruppamenti di atomi e sistemi macromolecolari.
Il metodo è caratterizzato da due fasi, combinando un algoritmo di salto globale per l'esplorazione di diversi bacini con una minimizzazione locale a ciascun salto. Il meccanismo di salto (*hopping*) tra bacini è conseguito mediante un approccio di tipo Monte Carlo.

I parametri di ingresso non opzionali della funzione `basinhopping` sono la funzione e il punto di partenza. Con la stessa scelta di x_0 di prima ($x_0 = 0.5$), vediamo che la funzione non riesce a trovare il minimo globale, rimanendo nel bacino iniziale e trovando lo stesso minimo locale di prima:

In [11]:
```
basinhopping(f,0.5)
```

Out[11]:
```
                     message: ['requested number of basinhopping iterations comp
leted successfully']
                     success: True
                         fun: -4.833586647031791
                           x: [ 8.497e-01]
                         nit: 100
      minimization_failures: 0
                        nfev: 1284
                        njev: 642
 lowest_optimization_result:  message: Optimization terminated successfully.
                              success: True
                               status: 0
                                  fun: -4.833586647031791
                                    x: [ 8.497e-01]
                                  nit: 4
                                  jac: [ 1.192e-07]
                             hess_inv: [[ 5.914e-02]]
                                 nfev: 12
                                 njev: 6
```

Usando $x_0 = 4.5$, vediamo che si perviene all'altro minimo locale.

In [12]:
```
basinhopping(f,4.5)
```

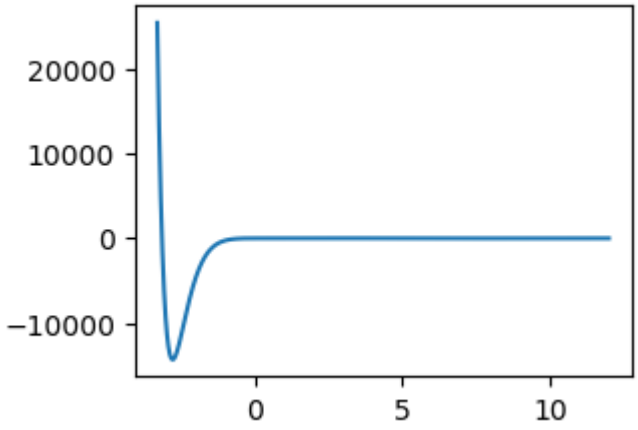
Out[12]:
```
                     message: ['requested number of basinhopping iterations comp
leted successfully']
                     success: True
                         fun: -7.9798802480396835
                           x: [ 8.806e+00]
                         nit: 100
      minimization_failures: 0
                        nfev: 956
                        njev: 478
 lowest_optimization_result:  message: Optimization terminated successfully.
                              success: True
                               status: 0
                                  fun: -7.9798802480396835
                                    x: [ 8.806e+00]
                                  nit: 3
                                  jac: [ 5.960e-08]
                              hess_inv: [[ 3.998e-01]]
                                 nfev: 10
                                 njev: 5
```

Tuttavia, con $x_0 = 3.5$, la funzione riesce a trovare il minimo assoluto, che non si vede nella figura di sopra ed è mostrato sotto (si noti che la scala dell'asse delle ordinate nella figura di sotto è tale da non far vedere i due minimi locali evidenziati sopra).

In [13]:
```
basinhopping(f,3.5)
```

Out[13]:
```
                     message: ['requested number of basinhopping iterations comp
leted successfully']
                     success: True
                         fun: -14358.318512503909
                           x: [-2.777e+00]
                         nit: 100
      minimization_failures: 50
                        nfev: 4085
                        njev: 1884
 lowest_optimization_result:  message: Optimization terminated successfully.
                              success: True
                               status: 0
                                  fun: -14358.318512503909
                                    x: [-2.777e+00]
                                  nit: 7
                                  jac: [ 0.000e+00]
                              hess_inv: [[ 9.777e-06]]
                                 nfev: 38
                                 njev: 19
```

In [14]:
```
x1 = np.linspace(-3.3,12,10**4)
y1 = f(x1)
plt.figure(figsize=(3.2,2.4))
plt.plot(x1,y1)
plt.show()
```



In [17]:
```
for a in (0.5,3.5,4.5):
    b = basinhopping(f,a,stepsize=1.0)
    print(b)
```

```
                     message: ['requested number of basinhopping iterations comple
ted successfully']
                     success: True
                         fun: -14358.318512503907
                           x: [-2.777e+00]
                         nit: 100
      minimization_failures: 12
                        nfev: 2058
                        njev: 986
 lowest_optimization_result:  message: Optimization terminated successfully.
                              success: True
                               status: 0
                                  fun: -14358.318512503907
                                    x: [-2.777e+00]
                                  nit: 9
                                  jac: [ 0.000e+00]
                              hess_inv: [[ 1.955e-06]]
                                 nfev: 32
                                 njev: 16
                     message: ['requested number of basinhopping iterations comple
ted successfully']
                     success: True
                         fun: -14358.318512503907
                           x: [-2.777e+00]
                         nit: 100
      minimization_failures: 47
                        nfev: 4013
                        njev: 1859
 lowest_optimization_result:  message: Optimization terminated successfully.
                              success: True
                               status: 0
                                  fun: -14358.318512503907
                                    x: [-2.777e+00]
                                  nit: 8
                                  jac: [ 0.000e+00]
                              hess_inv: [[ 8.800e-06]]
                                 nfev: 22
                                 njev: 11
                     message: ['requested number of basinhopping iterations comple
ted successfully']
                     success: True
                         fun: -7.9798802480396835
                           x: [ 8.806e+00]
                         nit: 100
      minimization_failures: 0
                        nfev: 1050
                        njev: 525
 lowest_optimization_result:  message: Optimization terminated successfully.
                              success: True
                               status: 0
                                  fun: -7.9798802480396835
                                    x: [ 8.806e+00]
                                  nit: 5
                                  jac: [ 5.960e-08]
                              hess_inv: [[ 3.997e-01]]
                                 nfev: 12
                                 njev: 6
```