Ottimizzazione dei Sistemi Complessi

G. Liuzzi[1]

Giovedì 22 Novembre 2018

[1] Istituto di Analisi dei Sistemi ed Informatica IASI - CNR

Algoritmo di campionamento uniforme

Se $\mathcal{A}$ è un intorno di un minimo globale x^* di f su $\mathcal{D}$, e si generano molti punti a caso su $\mathcal{D}$, allora è sempre più probabile che uno di questi punti cada in $\mathcal{A}$.

Questa considerazione è alla base del primo metodo di ottimizzazione globale che vediamo.

INPUT: maxit > 0

Genera $x_0 \in \mathcal{U}(\mathcal{D})$ e poni $f_{best} \leftarrow f(x_0)$ e $x_0^* \leftarrow x_0$

for $k = 1, \ldots,$maxit

Genera $x_k \in \mathcal{U}(\mathcal{D})$

end for

OUTPUT: $\{x_k^*\}$

Algoritmo di campionamento uniforme

Se $\mathcal{A}$ è un intorno di un minimo globale x^* di f su $\mathcal{D}$, e si generano molti punti a caso su $\mathcal{D}$, allora è sempre più probabile che uno di questi punti cada in $\mathcal{A}$.

Questa considerazione è alla base del primo metodo di ottimizzazione globale che vediamo.

INPUT: $\texttt{maxit} > 0$

Genera $x_0 \in \mathcal{U}(\mathcal{D})$ e poni $f_{best} \leftarrow f(x_0)$ e $x_0^* \leftarrow x_0$

for $k = 1, \ldots, \texttt{maxit}$

Genera $x_k \in \mathcal{U}(\mathcal{D})$

if $f(x_k) < f_{best}$ **then** $f_{best} \leftarrow f(x_k)$, $x_k^* \leftarrow x_k$

else poni $x_k^* \leftarrow x_{k-1}^*$

end for

OUTPUT: $\{x_k^*\}$

Algoritmo di campionamento uniforme

Se $\mathcal{A}$ è un intorno di un minimo globale x^* di f su $\mathcal{D}$, e si generano molti punti a caso su $\mathcal{D}$, allora è sempre più probabile che uno di questi punti cada in $\mathcal{A}$.

Questa considerazione è alla base del primo metodo di ottimizzazione globale che vediamo.

INPUT: `maxit` > 0

Genera $x_0 \in \mathcal{U}(\mathcal{D})$ e poni $f_{best} \leftarrow f(x_0)$ e $x_0^* \leftarrow x_0$

for $k = 1, \ldots,$`maxit`

  Genera $x_k \in \mathcal{U}(\mathcal{D})$

  if $f(x_k) < f_{best}$ **then** $f_{best} \leftarrow f(x_k)$, $x_k^* \leftarrow x_k$

  else poni $x_k^* \leftarrow x_{k-1}^*$

end for

OUTPUT: $\{x_k^*\}$

Generazione di punti a caso su $\mathcal{D}$

Il risultato fondamentale sul quale si basano molti metodi probabilistici è il seguente.

Proposizione

Sia $\{x_k\}$ una succ. di punti aleatori scelti a caso su $\mathcal{D}$. Allora, per ogni sottoinsieme $\mathcal{A} \subset \mathcal{D}$ tale che

$$meas(\mathcal{D}) > meas(\mathcal{A}) > 0$$

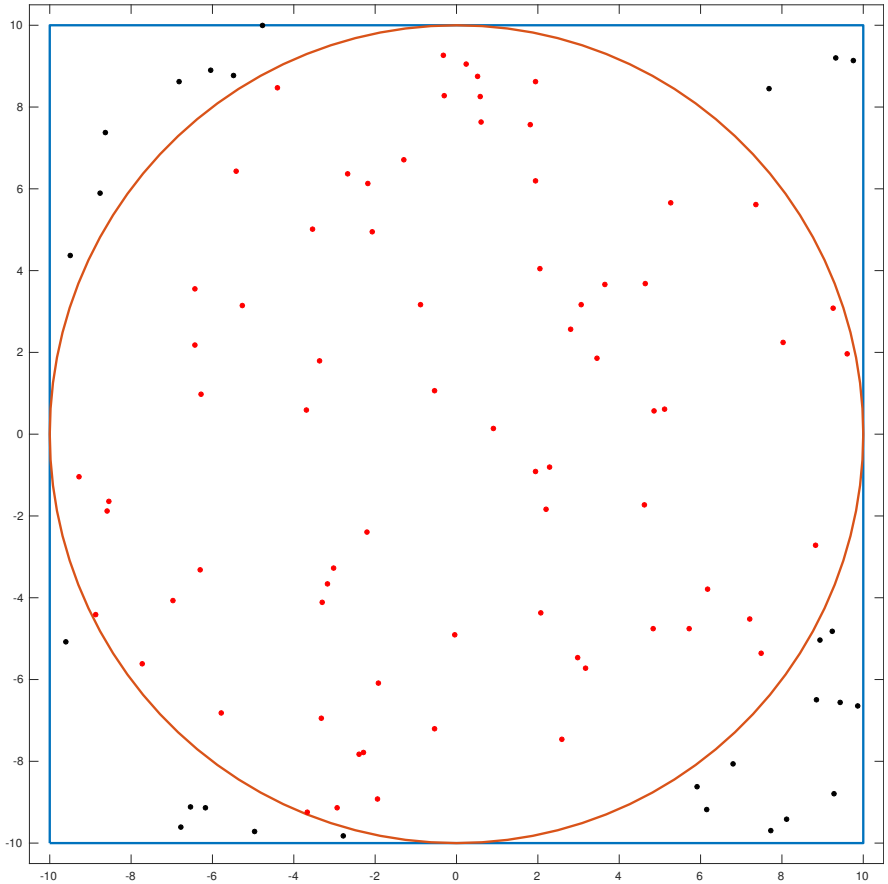
si ha

$$\lim_{k\to\infty} Prob\{X_k \cap \mathcal{A} \neq \emptyset\} = 1 \qquad con\ X_k = \{x_0, x_1, \ldots, x_k\}.$$

Esempio

100	2.920000e+02
1000	3.104000e+02
10000	3.136400e+02
100000	3.141760e+02

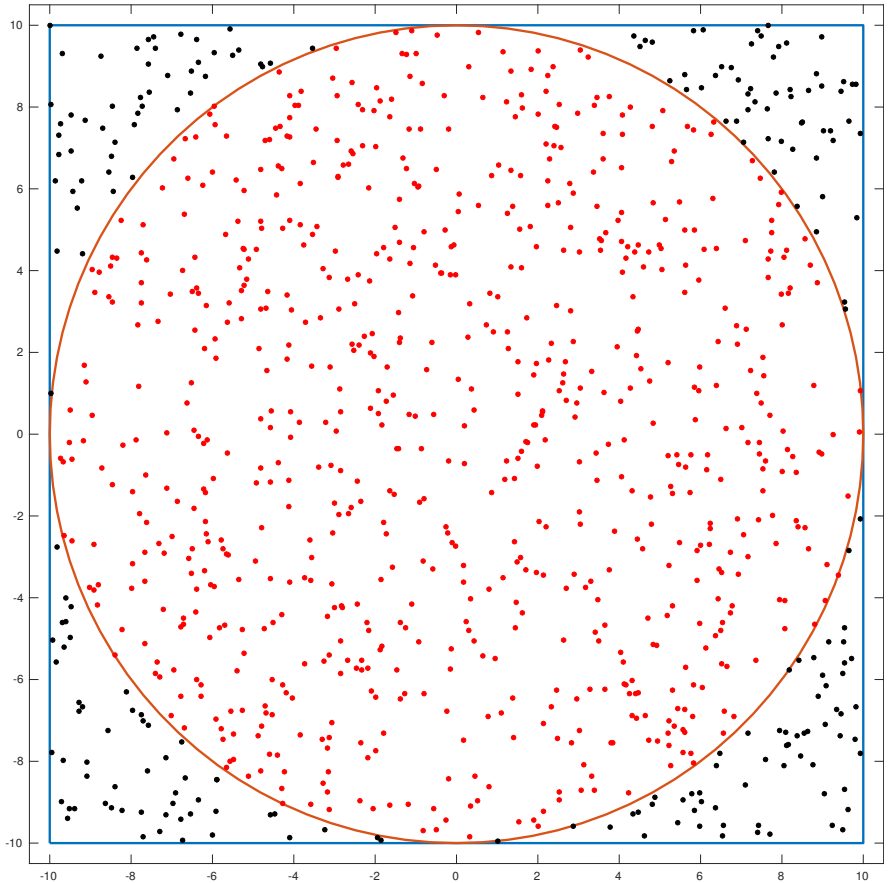
$\pi r^2 = 3.141593e+02$



Esempio

100	2.920000e+02
1000	3.104000e+02
10000	3.136400e+02
100000	3.141760e+02

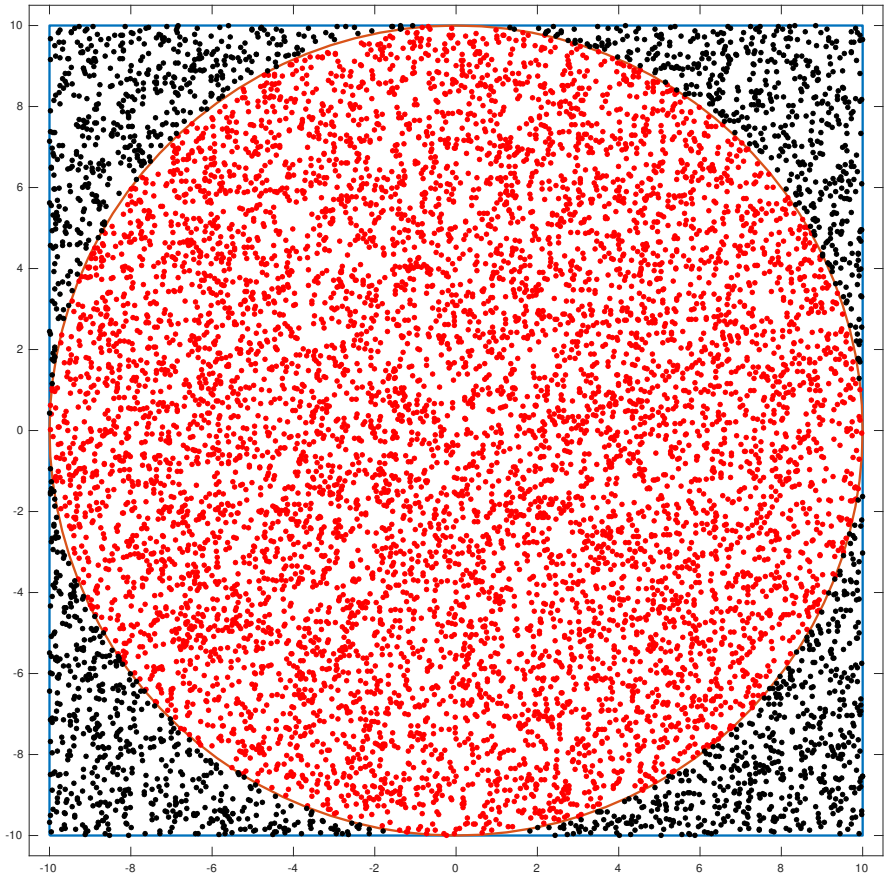
$\pi r^2 = 3.141593e+02$



Esempio

100	2.920000e+02
1000	3.104000e+02
10000	3.136400e+02
100000	3.141760e+02

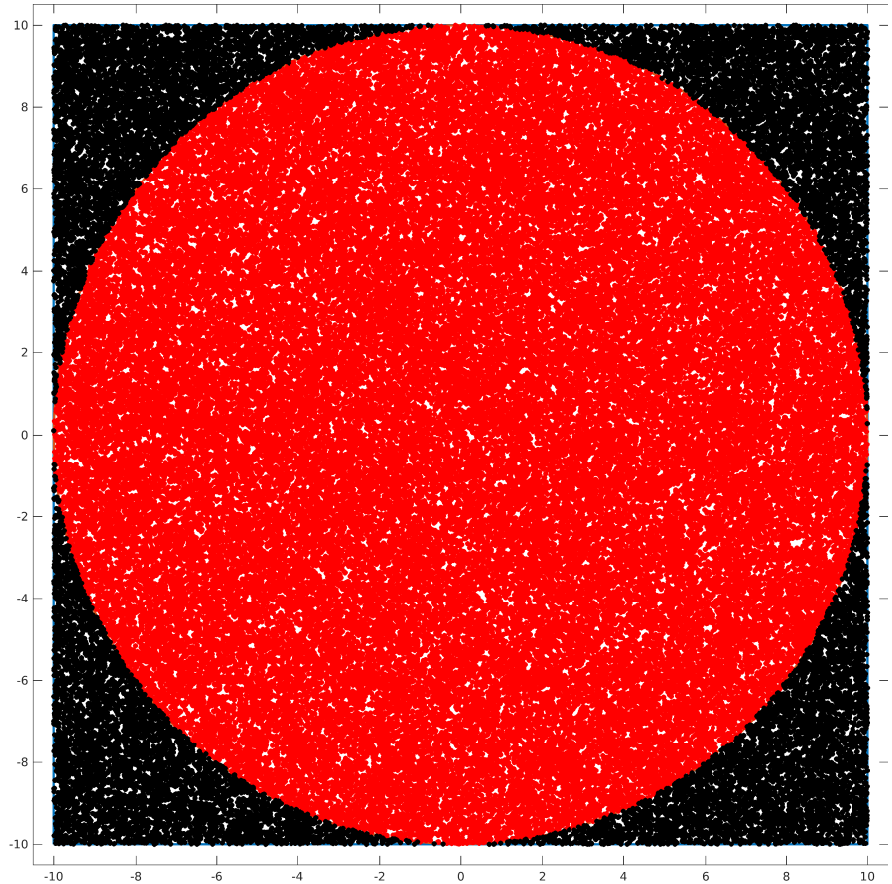
$$\pi r^2 = 3.141593e{+}02$$



Esempio

100	2.920000e+02
1000	3.104000e+02
10000	3.136400e+02
100000	3.141760e+02

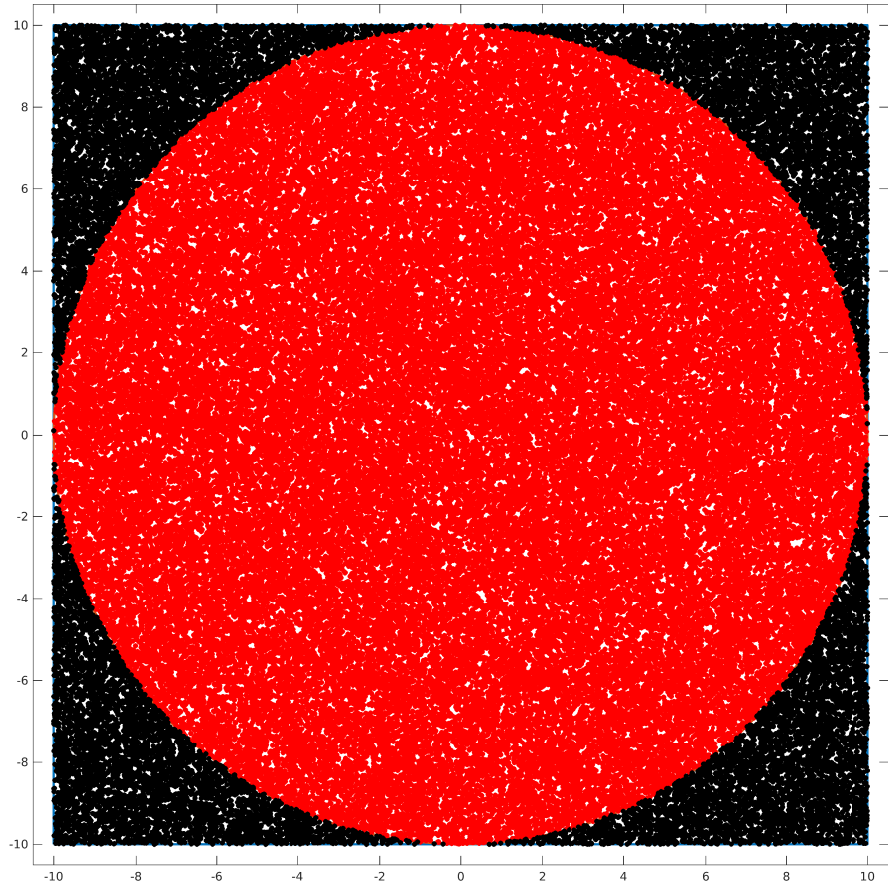
$\pi r^2 = 3.141593e+02$



Esempio

100	2.920000e+02
1000	3.104000e+02
10000	3.136400e+02
100000	3.141760e+02

$$\pi r^2 = 3.141593\text{e+}02$$



Notazione - minimizzazione locale

Sia $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}^n$ una mappa (che rappresenta un generico metodo di ottimizzazione locale) tale che:

1. per ogni $x \in \mathcal{D}$, $\bar{x} = \mathcal{L}(x)$ è un punto stazionario di f su $\mathcal{D}$ tale che $f(\bar{x}) \leq f(x)$;
2. per ogni minimo globale x^*, esiste un aperto $\mathcal{A}$ tale che se $x \in \mathcal{A}$, $x^* = \mathcal{L}(x)$.

Algoritmo di campionamento uniforme

INPUT: $\texttt{maxit} > 0$

Genera $x_0 \in \mathcal{D}$ e poni $f_{best} \leftarrow f(x_0)$ e $x_0^* \leftarrow x_0$

for $k = 0, \ldots, \texttt{maxit}$

Genera un punto $x_k \in \mathcal{D}$

if $f(x_k) < f_{best}$ **then** $f_{best} \leftarrow f(x_k)$, $x_k^* \leftarrow x_k$

else poni $x_k^* \leftarrow x_{k-1}^*$

end for

poni $x_{k+1}^* \leftarrow \mathcal{L}(x_k^*)$, $f_{best} \leftarrow f(x_{k+1}^*)$

OUTPUT: $\{x_k^*\}$, f_{best}

Algoritmo "best start"

INPUT: $\texttt{maxit} > 0$

Genera $x_0 \in \mathcal{D}$ e poni $f_{best} \leftarrow f(x_0)$ e $x_0^* \leftarrow x_0$

for $k = 0, \ldots, \texttt{maxit}$

    Genera un punto $x_k \in \mathcal{D}$
    if $f(x_k) < f_{best}$ **then** $f_{best} \leftarrow f(x_k)$, $x_k^* \leftarrow x_k$
    else poni $x_k^* \leftarrow x_{k-1}^*$

end for

poni $x_{k+1}^* \leftarrow \mathcal{L}(x_k^*)$, $f_{best} \leftarrow f(x_{k+1}^*)$

OUTPUT: $\{x_k^*\}$, f_{best}

Algoritmo "multi start"

INPUT: $\texttt{maxit} > 0$

Genera $x_0 \in \mathcal{D}$ e poni $x_0^* \leftarrow \mathcal{L}(x_0)$, $f_{best} \leftarrow f(x_0^*)$

for $k = 0, \ldots, \texttt{maxit}$

Genera un punto $x_k \in \mathcal{D}$, poni $\tilde{x}_k \leftarrow \mathcal{L}(x_k)$

if $f(\tilde{x}_k) < f_{best}$ **then** $f_{best} \leftarrow f(\tilde{x}_k)$, $x_k^* \leftarrow \tilde{x}_k$

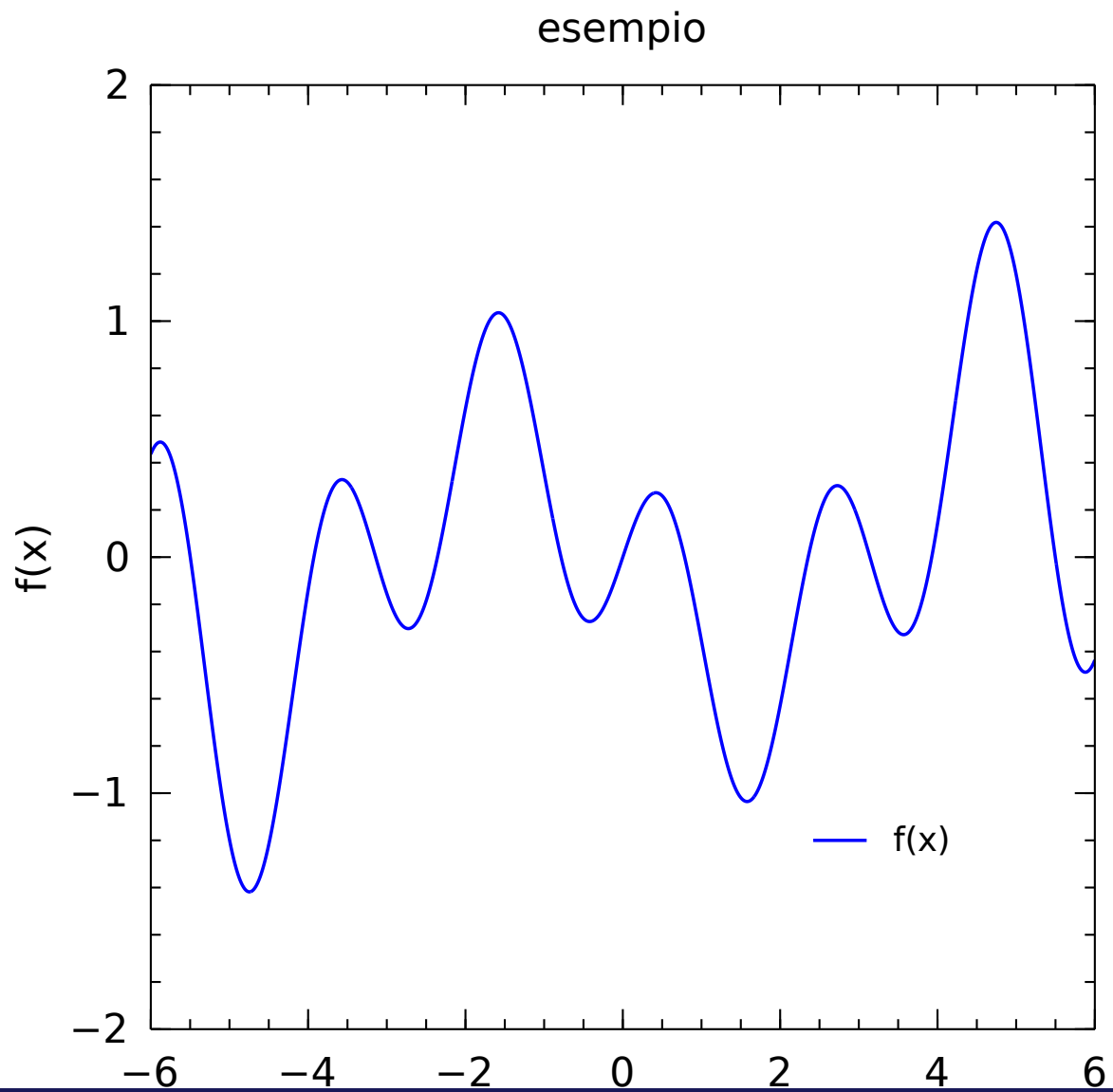
else poni $x_k^* \leftarrow x_{k-1}^*$

end for

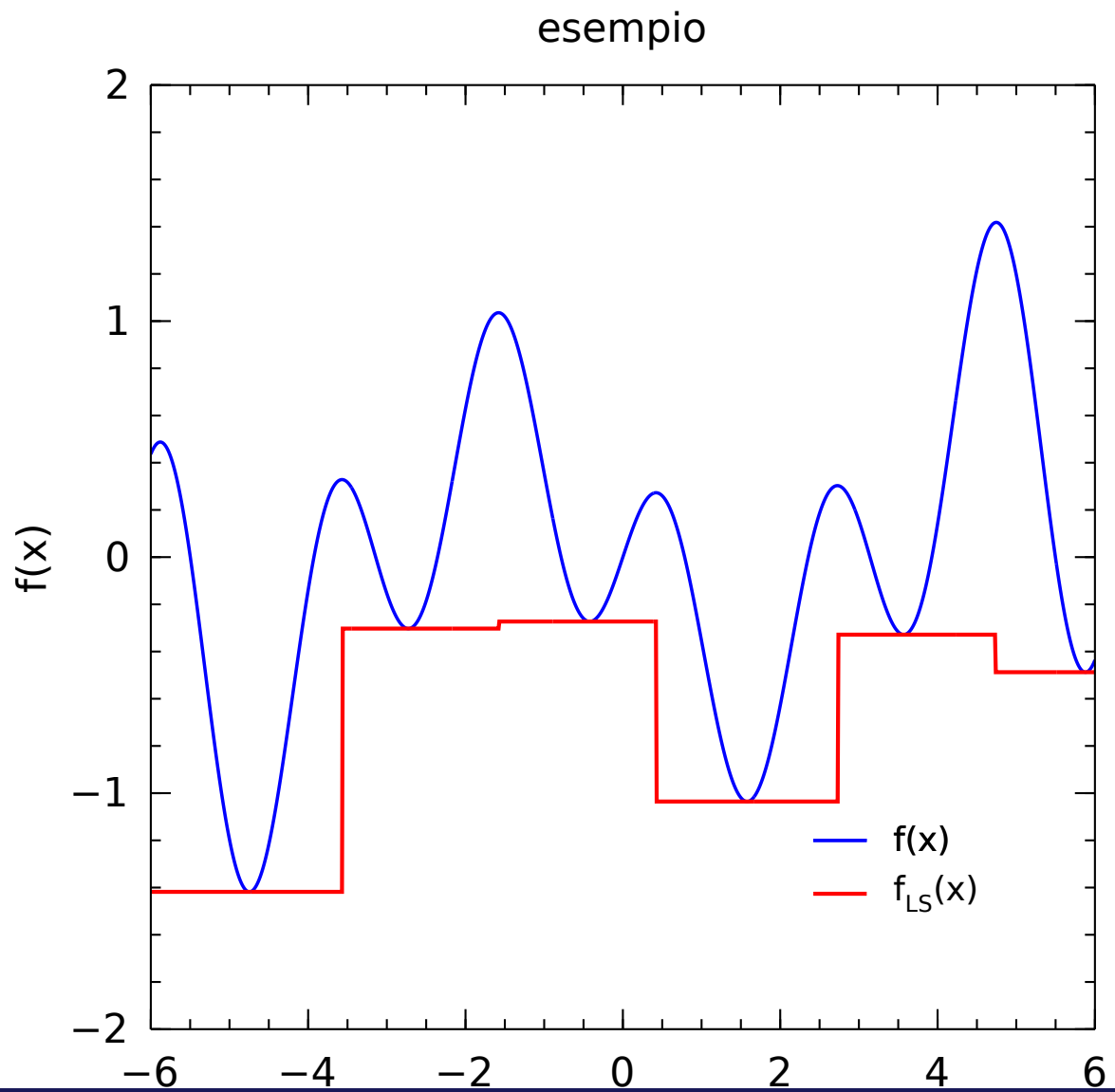
poni $x_{k+1}^* \leftarrow \mathcal{L}(x_k^*)$, $f_{best} \leftarrow f(x_{k+1}^*)$

OUTPUT: $\{x_k^*\}$, f_{best}

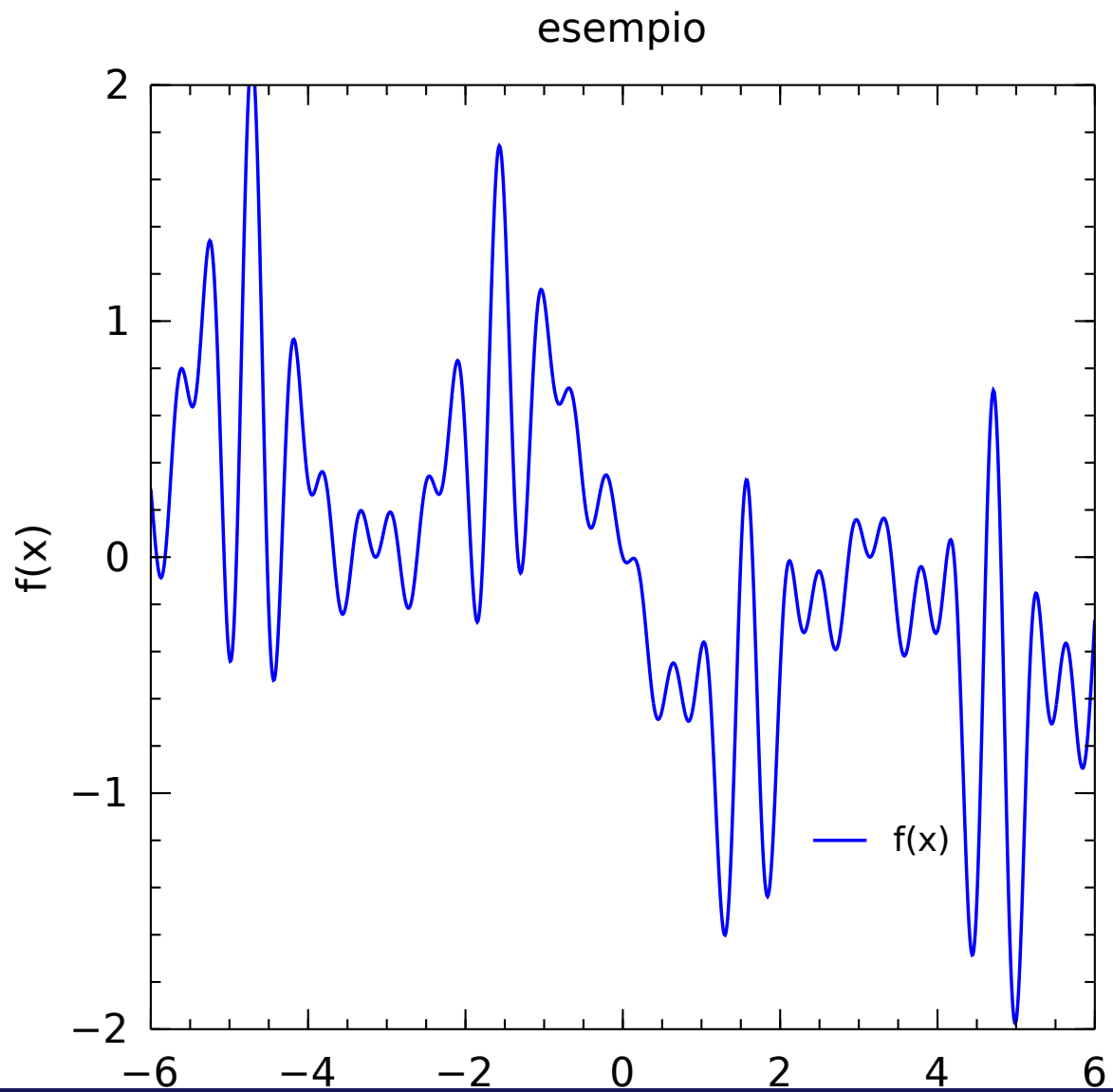
Funzione obiettivo modificata



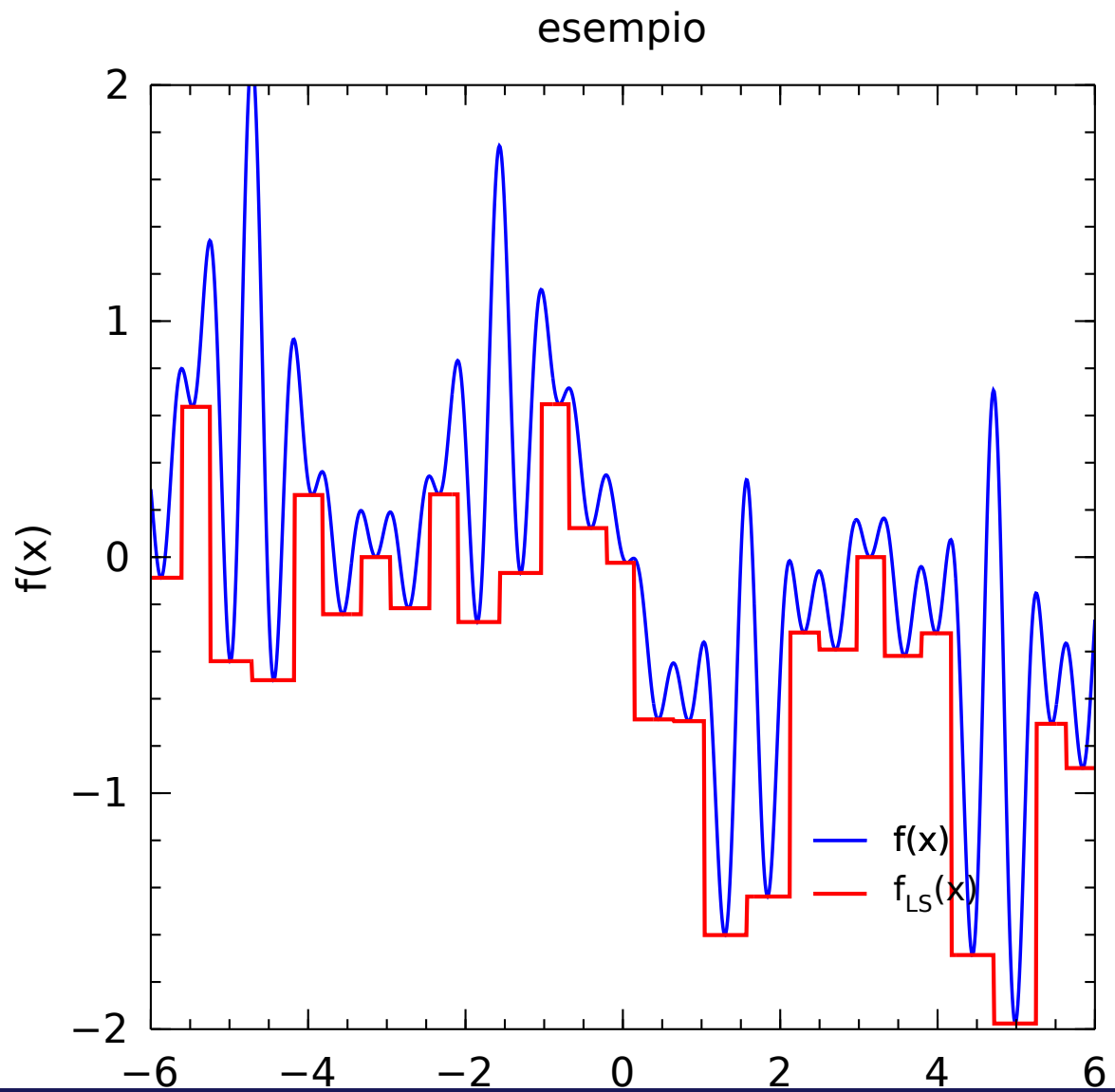
Funzione obiettivo modificata



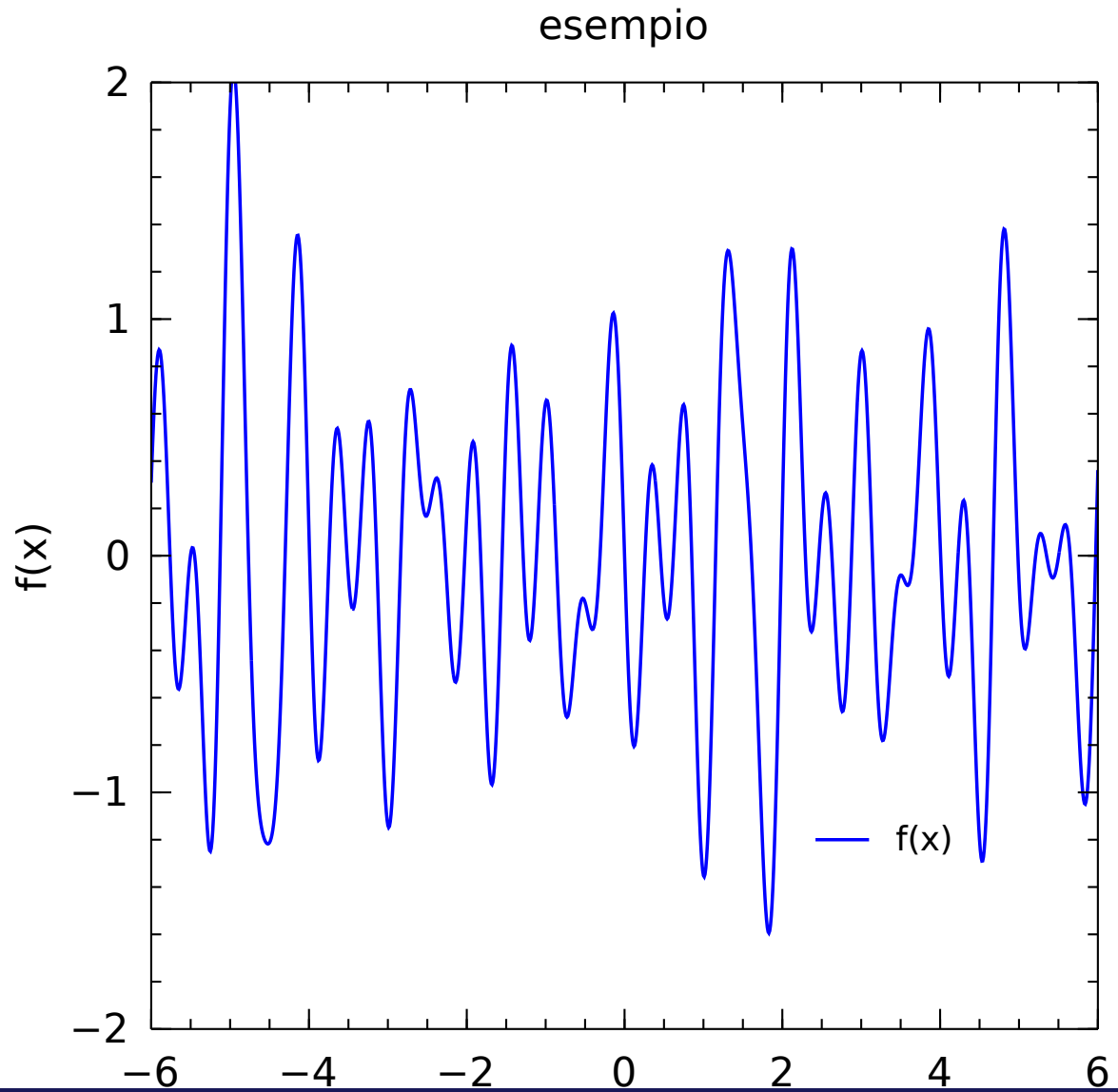
Funzione obiettivo modificata



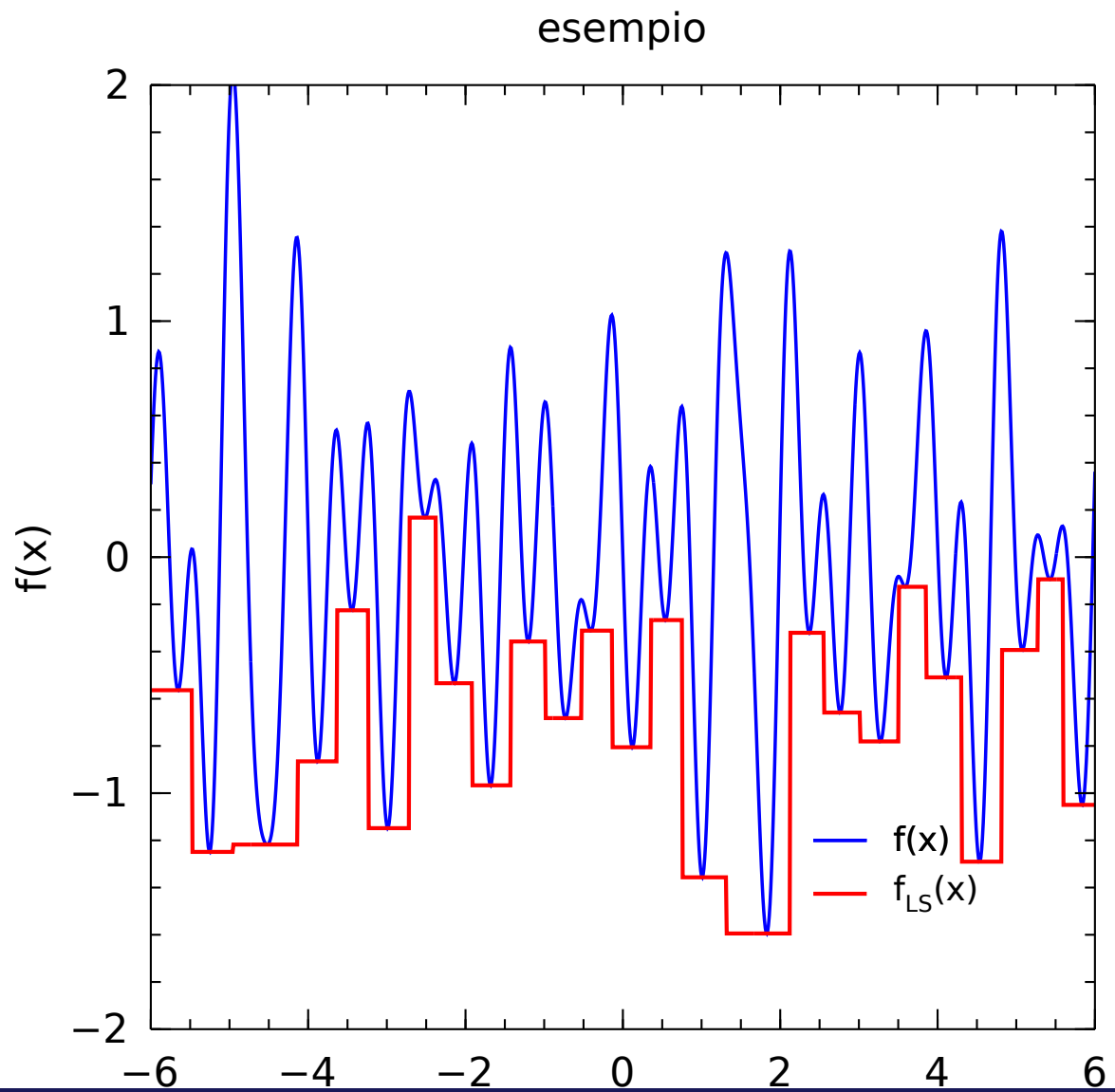
Funzione obiettivo modificata



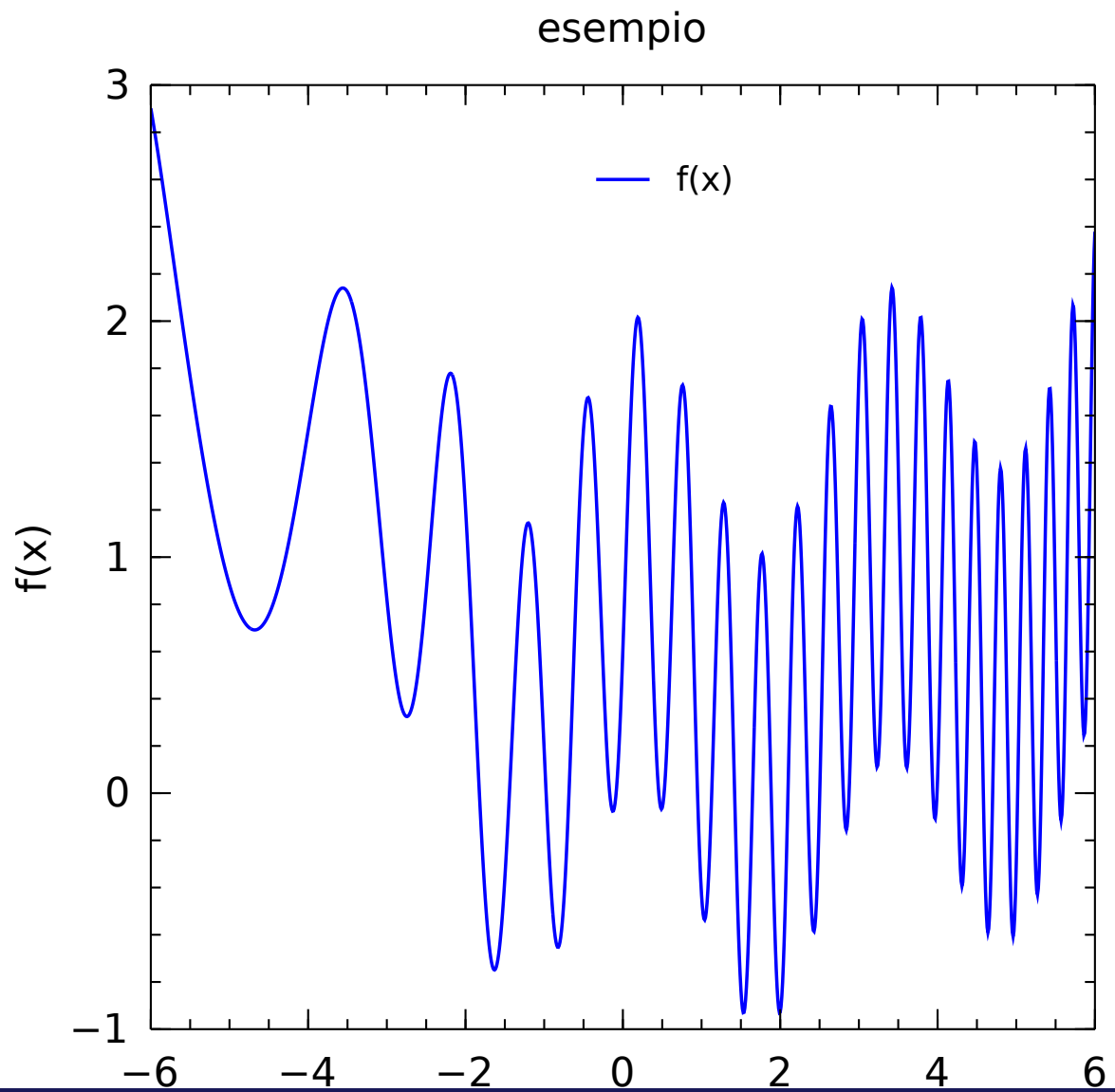
Funzione obiettivo modificata



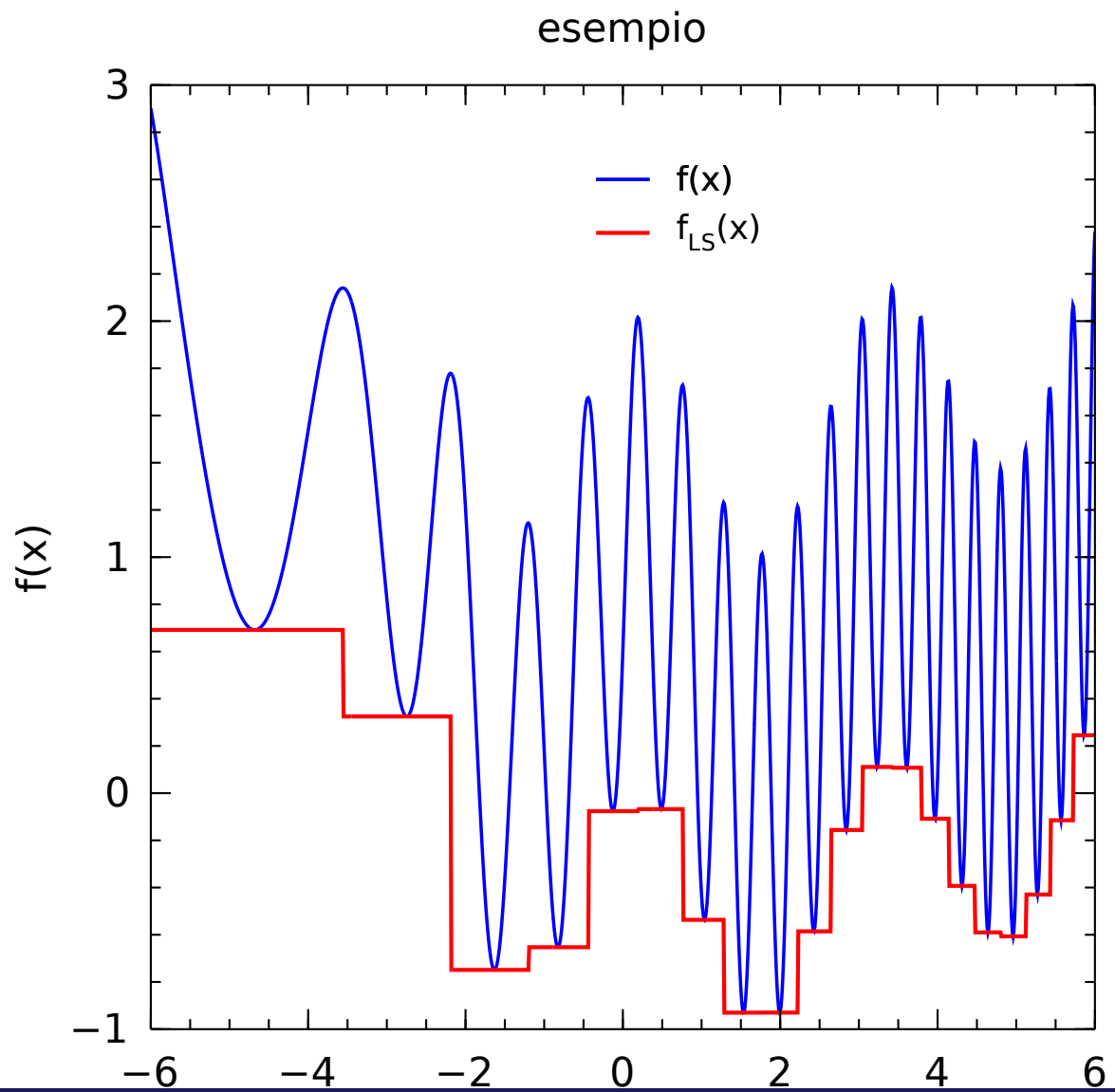
Funzione obiettivo modificata



Funzione obiettivo modificata



Funzione obiettivo modificata



Algoritmo di tipo "basin hopping"

È una variante "efficiente" del metodo multi-start.

L'idea è quella di migliorare volta per volta il minimo locale corrente generando il punto di partenza x_k in un intorno $\mathcal{N}(x_k^*)$ anziché su tutto il dominio $\mathcal{D}$.

Algoritmo "multi start"

INPUT: `maxit` > 0

Genera $x \in \mathcal{D}$ e poni $x^* \leftarrow \mathcal{L}(x)$, $f_{best} \leftarrow f(x^*)$

for $k = 0, \ldots,$`maxit`

Genera $x \in \mathcal{D}$, poni $\tilde{x} \leftarrow \mathcal{L}(x)$

if $f(\tilde{x}) < f_{best}$ **then** $f_{best} \leftarrow f(\tilde{x})$, $x^* \leftarrow \tilde{x}$

else *migliora localmente* $\tilde{x}$

while not criterio di arresto

Genera $x \in \mathcal{N}(\tilde{x})$, poni $y \leftarrow \mathcal{L}(x)$

if $f(y) < f(\tilde{x})$ **then** $\tilde{x} \leftarrow y$

end

if $f(\tilde{x}) < f_{best}$ **then** $f_{best} \leftarrow f(\tilde{x})$, $x^* \leftarrow \tilde{x}$

end

end for

OUTPUT: x^*, f_{best}

Algoritmo "basin hopping"

INPUT: $\texttt{maxit} > 0$

Genera $x \in \mathcal{D}$ e poni $x^* \leftarrow \mathcal{L}(x)$, $f_{best} \leftarrow f(x^*)$

for $k = 0, \ldots, \texttt{maxit}$

  Genera $x \in \mathcal{D}$, poni $\tilde{x} \leftarrow \mathcal{L}(x)$

  if $f(\tilde{x}) < f_{best}$ **then** $f_{best} \leftarrow f(\tilde{x})$, $x^* \leftarrow \tilde{x}$

  else *migliora localmente* $\tilde{x}$

    while not criterio di arresto

      Genera $x \in \mathcal{N}(\tilde{x})$, poni $y \leftarrow \mathcal{L}(x)$

      if $f(y) < f(\tilde{x})$ **then** $\tilde{x} \leftarrow y$

    end

    if $f(\tilde{x}) < f_{best}$ **then** $f_{best} \leftarrow f(\tilde{x})$, $x^* \leftarrow \tilde{x}$

  end

end for

OUTPUT: x^*, f_{best}

Algoritmo "basin hopping"

INPUT: $\texttt{maxit} > 0$

Genera $x \in \mathcal{D}$ e poni $x^* \leftarrow \mathcal{L}(x)$, $f_{best} \leftarrow f(x^*)$

for $k = 0, \ldots, \texttt{maxit}$

  Genera $x \in \mathcal{D}$, poni $\tilde{x} \leftarrow \mathcal{L}(x)$

  if $f(\tilde{x}) < f_{best}$ **then** $f_{best} \leftarrow f(\tilde{x})$, $x^* \leftarrow \tilde{x}$

  else *migliora localmente* $\tilde{x}$

    while not criterio di arresto

      Genera $x \in \mathcal{N}(\tilde{x})$, poni $y \leftarrow \mathcal{L}(x)$

      if $f(y) < f(\tilde{x})$ **then** $\tilde{x} \leftarrow y$

    end

    if $f(\tilde{x}) < f_{best}$ **then** $f_{best} \leftarrow f(\tilde{x})$, $x^* \leftarrow \tilde{x}$

  end

end for

OUTPUT: x^*, f_{best}

Algoritmo basin hopping

INPUT: $\texttt{maxit} > 0$

Genera $x \in \mathcal{D}$ e poni $x^* \leftarrow \mathcal{L}(x)$, $f_{best} \leftarrow f(x^*)$

for $k = 0, \ldots, \texttt{maxit}$

Genera $x \in \mathcal{D}$, poni $\tilde{x} \leftarrow \mathcal{L}(x)$

migliora localmente $\tilde{x}$

while not criterio di arresto

Genera $x \in \mathcal{N}(\tilde{x})$, poni $y \leftarrow \mathcal{L}(x)$

if $f(y) < f(\tilde{x})$ **then** $\tilde{x} \leftarrow y$

end

if $f(\tilde{x}) < f_{best}$ **then** $f_{best} \leftarrow f(\tilde{x})$, $x^* \leftarrow \tilde{x}$

end

end for

OUTPUT: x^*, f_{best}

Algoritmo basin hopping

INPUT: `maxit` > 0

Genera $x \in \mathcal{D}$ e poni $x^* \leftarrow \mathcal{L}(x)$, $f_{best} \leftarrow f(x^*)$

while not criterio di arresto globale

  Genera $x \in \mathcal{D}$, poni $\tilde{x} \leftarrow \mathcal{L}(x)$

  migliora localmente $\tilde{x}$

  while not criterio di arresto locale

    Genera $x \in \mathcal{N}(\tilde{x})$, poni $y \leftarrow \mathcal{L}(x)$

    if $f(y) < f(\tilde{x})$ **then** $\tilde{x} \leftarrow y$

  end

  if $f(\tilde{x}) < f_{best}$ **then** $f_{best} \leftarrow f(\tilde{x})$, $x^* \leftarrow \tilde{x}$

  end

end for

OUTPUT: x^*, f_{best}

Algoritmo basin hopping monotono

INPUT: $\texttt{maxit} > 0$

Genera $x \in \mathcal{D}$ e poni $x^* \leftarrow \mathcal{L}(x)$, $f_{best} \leftarrow f(x^*)$

while not criterio di arresto globale

  Genera $x \in \mathcal{D}$, poni $\tilde{x} \leftarrow \mathcal{L}(x)$

  $\boxed{\textit{migliora localmente } \tilde{x}}$

  while not criterio di arresto locale

    Genera $x \in \mathcal{N}(\tilde{x})$, poni $y \leftarrow \mathcal{L}(x)$

    if $\boxed{f(y) < f(\tilde{x})}$ **then** $\tilde{x} \leftarrow y$

  end

  if $f(\tilde{x}) < f_{best}$ **then** $f_{best} \leftarrow f(\tilde{x})$, $x^* \leftarrow \tilde{x}$

  end

end for

OUTPUT: x^*, f_{best}

Algoritmo basin hopping nonmonotono

Si sostituisce il criterio

$$f(y) < f(\tilde{x})$$

con il criterio

$$\mathcal{U}(0,1) \leq \exp(-(f(y) - f(\tilde{x}))/T)$$

- quando $f(y) < f(\tilde{x})$, $\exp(-(f(y) - f(\tilde{x}))/T) > 1$ quindi il criterio è banalmente soddisfatto
- quando $f(y) \geq f(\tilde{x})$, $\exp(-(f(y) - f(\tilde{x}))/T) \leq 1$ quindi
  - se $\mathcal{U}(0,1) > \exp(-(f(y) - f(\tilde{x}))/T)$ il punto proposto è "rifiutato"
  - se $\mathcal{U}(0,1) \leq \exp(-(f(y) - f(\tilde{x}))/T)$ il punto proposto è "accettato" anche se è un punto peggiore rispetto a $\tilde{x}$

Algoritmo basin hopping nonmonotono

INPUT: $\texttt{maxit} > 0$

Genera $x \in \mathcal{D}$ e poni $x^* \leftarrow \mathcal{L}(x)$, $f_{best} \leftarrow f(x^*)$

while not criterio di arresto globale

  Genera $x \in \mathcal{D}$, poni $\tilde{x} \leftarrow \mathcal{L}(x)$

  $\boxed{\textit{migliora localmente } \tilde{x}}$

  while not criterio di arresto locale

    Genera $x \in \mathcal{N}(\tilde{x})$, poni $y \leftarrow \mathcal{L}(x)$

    if $\boxed{\mathcal{U}(0,1) \leq \exp(-(f(y) - f(\tilde{x}))/T)}$ **then** $\tilde{x} \leftarrow y$

  end

  if $f(\tilde{x}) < f_{best}$ **then** $f_{best} \leftarrow f(\tilde{x})$, $x^* \leftarrow \tilde{x}$

  end

end for

OUTPUT: x^*, f_{best}

Basin hopping – crit. di arresto locale

Un criterio molto usato consiste nell'imporre un massimo numero di iterazioni in cui non si aggiorna $\tilde{x}$

Il ciclo interno diventa:

$h \leftarrow 0$

while $h \leq$ `max_no_improve`

Genera $x \in \mathcal{N}(\tilde{x})$, poni $y \leftarrow \mathcal{L}(x)$

if $y \prec x$ **then** $\tilde{x} \leftarrow y$, $h \leftarrow 0$

else $h \leftarrow h + 1$

end

Basin hopping – crit. di arresto locale

Un criterio molto usato consiste nell'imporre un massimo numero di iterazioni in cui non si aggiorna $\tilde{x}$

Il ciclo interno diventa:

$h \leftarrow 0$

while $h \leq$ `max_no_improve`

  Genera $x \in \mathcal{N}(\tilde{x})$, poni $y \leftarrow \mathcal{L}(x)$

  if $y \prec x$ **then** $\tilde{x} \leftarrow y$, $h \leftarrow 0$

  else $h \leftarrow h + 1$

end

Basin hopping – crit. di arresto locale

Un criterio molto usato consiste nell'imporre un massimo numero di iterazioni in cui non si aggiorna $\tilde{x}$

Il ciclo interno diventa:

$h \leftarrow 0$

while $h \leq$ `max_no_improve`

Genera $x \in \mathcal{N}(\tilde{x})$, poni $y \leftarrow \mathcal{L}(x)$

if $y \prec x$ **then** $\tilde{x} \leftarrow y$, $h \leftarrow 0$

else $h \leftarrow h + 1$

end

Algoritmo Controlled Random Search (CRS)

Algoritmo in due fasi

1. Fase Globale: generazione random di m punti su $\mathcal{D}$
2. Face Locale: miglioramento iterativo degli m punti

Algoritmo Controlled Random Search (CRS)

Algoritmo in due fasi

1. Fase Globale: generazione random di m punti su $\mathcal{D}$
2. Face Locale: miglioramento iterativo degli m punti

Algoritmo Controlled Random Search (CRS)

Algoritmo in due fasi

1. Fase Globale: generazione random di m punti su $\mathcal{D}$
2. Face Locale: miglioramento iterativo degli m punti

Algoritmo Controlled Random Search (CRS)

INPUT: $m \gg n$, `maxit`, `tol`

(*fase globale*) Genera $S_0 = \{x_i \in \mathcal{U}(\mathcal{D}),\ i = 1, \ldots, m\}$

$k \leftarrow 0$

Calcola $f_{\min} = \min_{x_i \in S_0} f(x_i)$, $f_{\max} = \max_{x_i \in S_0} f(x_i)$

while $k \leq$`maxit` **and** $f_{\max} - f_{\min} >$`tol`

$k \leftarrow k + 1$

end

Algoritmo Controlled Random Search (CRS)

INPUT: $m \gg n$, `maxit`, `tol`

(*fase globale*) Genera $S_0 = \{x_i \in \mathcal{U}(\mathcal{D}),\ i = 1, \ldots, m\}$

$k \leftarrow 0$

Calcola $f_{\min} = \min_{x_i \in S_0} f(x_i)$, $f_{\max} = \max_{x_i \in S_0} f(x_i)$

while $k \leq$`maxit` **and** $f_{\max} - f_{\min} >$`tol`

Seleziona a caso $X_k \subset S_k$ t.c. $|X_k| = n + 1$

Seleziona a caso $y \in X_k$

$k \leftarrow k + 1$

end

Algoritmo Controlled Random Search (CRS)

INPUT: $m \gg n$, `maxit`, `tol`

(*fase globale*) Genera $S_0 = \{x_i \in \mathcal{U}(\mathcal{D}),\ i = 1, \ldots, m\}$

$k \leftarrow 0$

Calcola $f_{\min} = \min_{x_i \in S_0} f(x_i)$, $f_{\max} = \max_{x_i \in S_0} f(x_i)$

while $k \leq$`maxit` **and** $f_{\max} - f_{\min} >$`tol`

Seleziona a caso $X_k \subset S_k$ t.c. $|X_k| = n + 1$

Seleziona a caso $y \in X_k$

Calcola $\bar{x} \leftarrow \sum_{x \in X_k, x \neq y} x/n$

Calcola $z \leftarrow 2\bar{x} - y$

$k \leftarrow k + 1$

end

Algoritmo Controlled Random Search (CRS)

INPUT: $m \gg n$, `maxit`, `tol`

(*fase globale*) Genera $S_0 = \{x_i \in \mathcal{U}(\mathcal{D}),\ i = 1, \ldots, m\}$

$k \leftarrow 0$

Calcola $f_{\min} = \min_{x_i \in S_0} f(x_i)$, $f_{\max} = \max_{x_i \in S_0} f(x_i)$

while $k \leq$`maxit` **and** $f_{\max} - f_{\min} >$`tol`

Seleziona a caso $X_k \subset S_k$ t.c. $|X_k| = n + 1$

Seleziona a caso $y \in X_k$

Calcola $\bar{x} \leftarrow \sum_{x \in X_k, x \neq y} x/n$

Calcola $z \leftarrow 2\bar{x} - y$

if $z \in \mathcal{D}$ and $f(z) < f_{\max}$

then $S_{k+1} \leftarrow S_k \setminus \{x_{\max}\} \cup \{z\}$

else $S_{k+1} \leftarrow S_k$

$k \leftarrow k + 1$

end

Algoritmo Controlled Random Search (CRS)

INPUT: $m \gg n$, `maxit`, `tol`

(*fase globale*) Genera $S_0 = \{x_i \in \mathcal{U}(\mathcal{D}),\ i = 1, \ldots, m\}$

$k \leftarrow 0$

Calcola $f_{\min} = \min_{x_i \in S_0} f(x_i)$, $f_{\max} = \max_{x_i \in S_0} f(x_i)$

while $k \leq$`maxit` **and** $f_{\max} - f_{\min} >$`tol`

Seleziona a caso $X_k \subset S_k$ t.c. $|X_k| = n + 1$

Seleziona a caso $y \in X_k$

Calcola $\bar{x} \leftarrow \sum_{x \in X_k, x \neq y} x/n$

Calcola $z \leftarrow 2\bar{x} - y$

if $z \in \mathcal{D}$ **and** $f(z) < f_{\max}$

then $S_{k+1} \leftarrow S_k \setminus \{x_{\max}\} \cup \{z\}$

else $S_{k+1} \leftarrow S_k$

$k \leftarrow k + 1$

end

Algoritmo Controlled Random Search (CRS)

INPUT: $m \gg n$, `maxit`, `tol`

(*fase globale*) Genera $S_0 = \{x_i \in \mathcal{U}(\mathcal{D}),\ i = 1, \ldots, m\}$

$k \leftarrow 0$

Calcola $f_{\min} = \min_{x_i \in S_0} f(x_i)$, $f_{\max} = \max_{x_i \in S_0} f(x_i)$

while $k \leq$`maxit` **and** $f_{\max} - f_{\min} >$`tol`

Seleziona a caso $X_k \subset S_k$ t.c. $|X_k| = n + 1$

Seleziona a caso $y \in X_k$

Calcola $\bar{x} \leftarrow \sum_{x \in X_k, x \neq y} x/n$

Calcola $z \leftarrow 2\bar{x} - y$

if $z \in \mathcal{D}$ **and** $f(z) < f_{\max}$

then $S_{k+1} \leftarrow S_k \setminus \{x_{\max}\} \cup \{z\}$

else $S_{k+1} \leftarrow S_k$

$k \leftarrow k + 1$

end