**Universidade de Fortaleza - UNIFOR**

# Sistemas Inteligentes - T296

**Msc. Prof. Paulo Cirillo Souza Barbosa**
Centro de Ciências Tecnológicas - CCT
Universidade de Fortaleza
Fortaleza, Ceará, Brasil

# Sumário

1. Otimização.
2. Introdução aos Algoritmos Evolucionários.

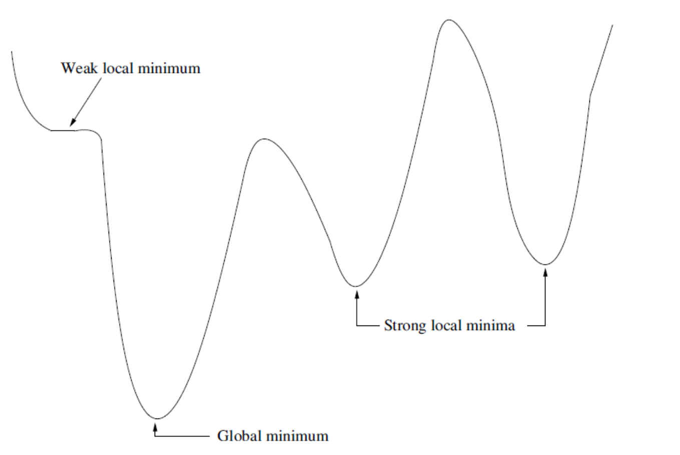
## Otimização e Busca - Problemática.

- A **otimização** é uma área da matemática aplicada que possui o foco no estudo de métodos de resolução de problemas em que se procura **minimizar** ou **maximizar** uma função numérica.
- Tal processo é realizado pela escolha sistemática dos valores de certas variáveis comumente conhecidas como variáveis de decisão.
- A **busca** pode ser vista como uma metodologia de resolução de problemas que toma como base a sua formulação em um **espaço de estados** e um elemento neste espaço é visto como uma solução para o problema.
- Tendo em vista que nem toda solução tem a mesma qualidade, busca-se encontrar a **ótima** para o problema.
- Exemplos: Projeto de circuitos integrados, escalonamento de jornadas de trabalho, arranjo físico de maquinário em indústria, otimização de rede de telecomunicações, roteamento de veículos.

## Otimização e Busca - Formalização do problema de otimização.

- Um problema de otimização geralmente apresentam três componentes básicas:
  1. Uma **função objetivo/custo** $f : \mathbb{R}^p \to \mathbb{R}$, que é o critério que deseja-se otimizar. Ou seja, a quantidade a ser minimizada ou maximizada.
  2. O **conjunto de de variáveis de decisão** $\mathbf{x} \in \mathbb{R}^p$, que afetam diretamente a função objetivo. Considerando $\mathbf{x}$ como variável independente, então $f(\mathbf{x})$ quantifica a qualidade da solução candidata $\mathbf{x}$
  3. Um **conjunto de restrições**, que limita os valores que podem ser atribuídos às variáveis independentes.
- Além disso, outros conceitos importantes como:
  1. **Espaço de estados** $(S)$.
  2. **Vizinhança*** $(V)$: Dado um ponto $\mathbf{x} \in S$, $V(\mathbf{x})$ representa todos os pontos $\mathbf{y} \in S$ que satisfazem $|\mathbf{x} - \mathbf{y}| \leq \varepsilon$.
  3. **Ótimo local ($\mathbf{x}_l*$):** $\mathbf{x}_l*$ é um mínimo/máximo local se $\forall \mathbf{x} \in V(\mathbf{x}_l*)$, $F(\mathbf{x}_l*) \leq F(\mathbf{x})$ (ou para máximo: $F(\mathbf{x}_l*) \geq F(\mathbf{x})$
  4. **Ótimo global ($\mathbf{x}_g*$):** $\mathbf{x}_g*$ é um mínimo/máximo global se $\forall \mathbf{x} \in S$, $F(\mathbf{x}_g*) \leq F(\mathbf{x})$ (ou para máximo: $F(\mathbf{x}_g*) \geq F(\mathbf{x})$)

## Otimização e Busca.



## Otimização e Busca.

- Um problema de otimização, pode ser categorizado por:
  1. A quantidade de variáveis: problema univariado ou multivariado.
  2. O tipo da variável: problema de domínio contínuo, discreto, misto ou até por permutações de inteiros (combinatória).
  3. Grau de não-linearidade da função objetivo.
  4. Restrições utilizadas: que definem a restrição no espaço de estados (neste caso o espaço é reduzido para $\mathbb{F}$).
  5. Quantidade de soluções ótimas: unimodal $\times$ multimodal.
  6. Quantidade de critérios de otimização.

## Otimização e Busca.

- Um **algoritmo** de otimização, busca uma solução ótima por iterações que realizam uma pertubação de uma solução ótima corrente, na esperança de encontrar uma nova solução ótima.
- Os algoritmos também possuem suas categorias: local, global, determinístico ou estocástico.
- Um problema de otimização sem restrição, pode ser escrito da seguinte forma:

$$\begin{gathered} minimize\ f(x), \mathbf{x} = (x_1, x_2, \cdots, x_p) \\ subject\ to\ x_j \in dom(x_j) \end{gathered}$$

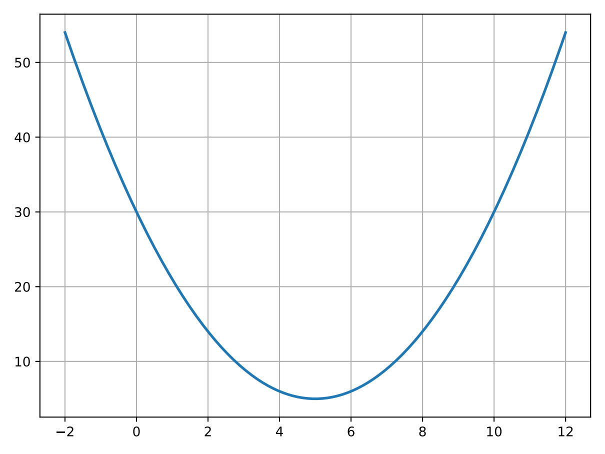
em que $dom(x_j)$ é o domínio da variável $x_j$ e para um problema contínuo, este domínio para cada variável é o conjunto dos reais ($\mathbb{R}$).

## Otimização e Busca - **busca local**.

- O problema pode ter natureza **discreta** ou **contínua**;
- No caso contínuo, chamado também de otimização numérica, pode-se ter um número infinito de possíveis soluções.
- No caso discreto, também chamado de otimização combinatória, as soluções são frutos de uma certa combinação de parâmetros discretos e possuem um número finito de soluções.
- Exemplos: sintonização de controladores PID ou redes neurais.
- Solução do problema do Caixeiro Viajante ou problema das 8 Rainhas.

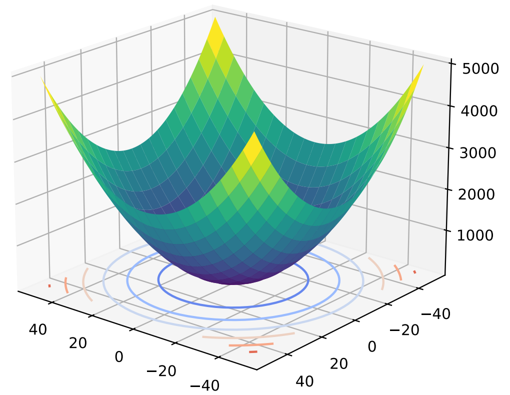
## Otimização e Busca - **busca local**.

- Exemplos de funções: unimodal × multimodal



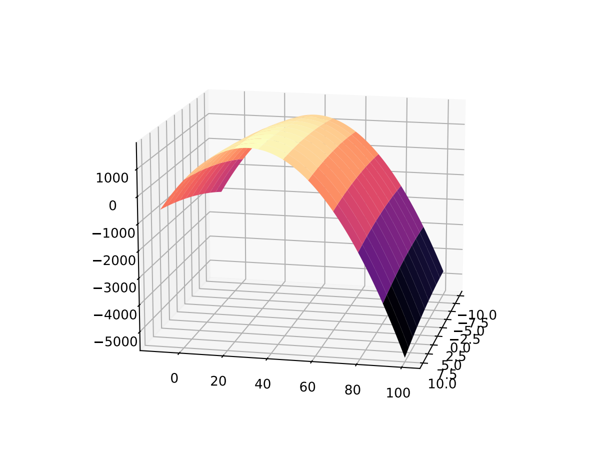
## Otimização e Busca - **busca local**.

- Exemplos de funções: unimodal × multimodal



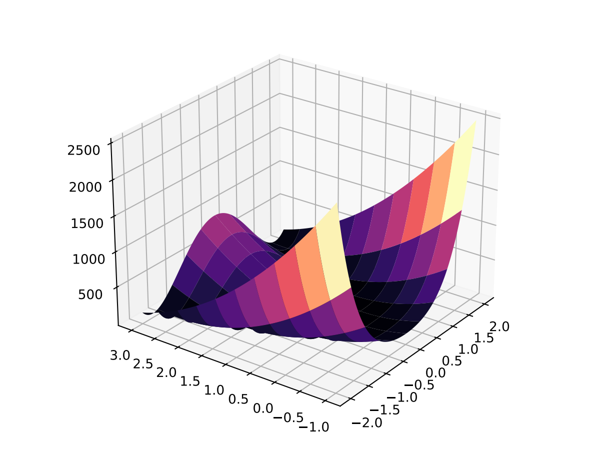
## Otimização e Busca - **busca local**.

- Exemplos de funções: unimodal × multimodal



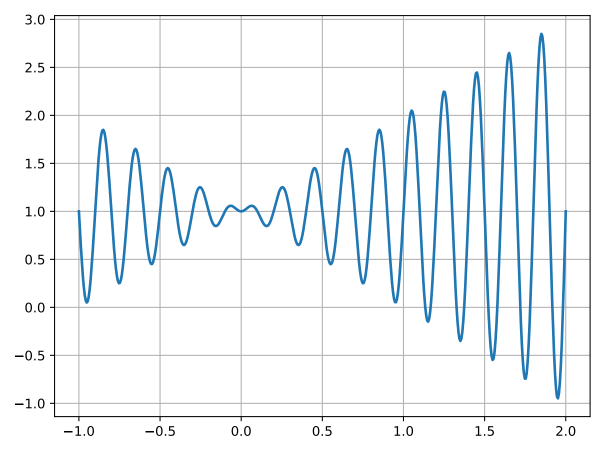
## Otimização e Busca - **busca local**.

- Exemplos de funções: unimodal $\times$ multimodal



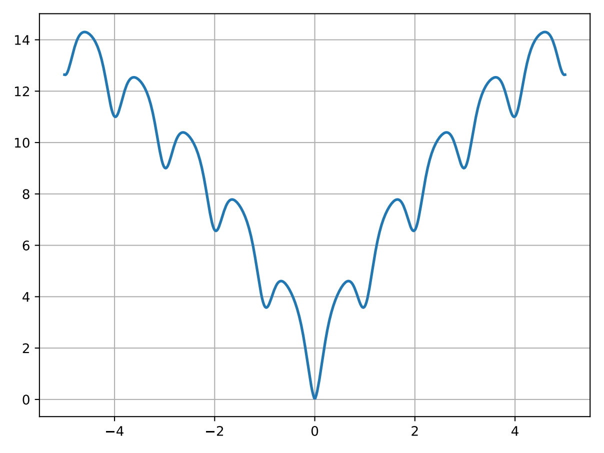
## Otimização e Busca - **busca local**.

- Exemplos de funções: unimodal $\times$ multimodal



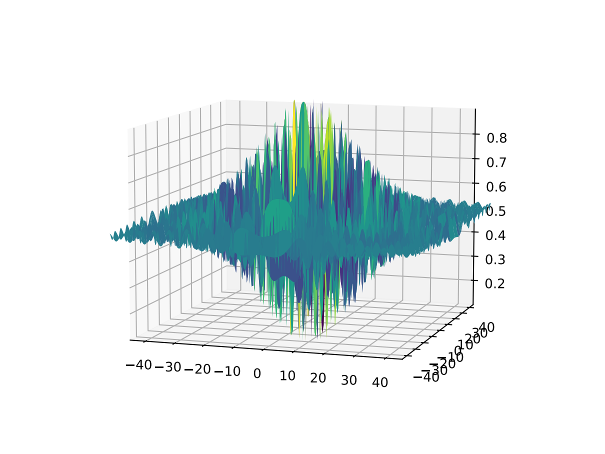
## Otimização e Busca - **busca local**.

- Exemplos de funções: unimodal $\times$ multimodal



## Otimização e Busca - **busca local**.

- Exemplos de funções: unimodal $\times$ multimodal



**Otimização e Busca - busca local Algoritmo da Subida de Encosta (*Hill Climbing*).**

- É um tipo de algoritmo de busca heurística local.
- A partir de um estado inicial, escolhe um sucessor melhor "subir sempre"até encontrar um pico.
- Para problemas convexos, este algoritmo encontra valores de ótimo global.
- Caso o problema seja não-convexo (multimodal), encontra apenas o ótimo local.
- O algoritmo é baseado nos seguintes passos:

## Otimização e Busca - **busca local** Algoritmo da Subida de Encosta (*Hill Climbing*).

**Algorithm 1:** Pseudocódigo busca por subida de encosta.

Inicializar o ponto inicial (zero ou limite de domínio) $x_0$
definir o valor $\varepsilon$ para candidato vizinho.
Definir uma quantidade máxima de iterações $max_{it}$ e quantidade máxima de candidatos (possíveis vizinhos) $max_n$.
Melhor valor $x_{best} \leftarrow x_0$ e melhor valor computado $f_{best} \leftarrow f(x_{best})$.
$i \leftarrow 0$
**while** $i < max_{it}$ E houver melhoria **do**
    $j \leftarrow 0$
    **while** $j < max_n$ **do**
        $j \leftarrow j + 1$
        melhoria $\leftarrow$ false
        $y \leftarrow candidato(x_{best})$.
        $F \leftarrow f(y)$
        **if** $F > f_{best}$ **then**
            $x_{best} \leftarrow y$
            $f_{best} \leftarrow F$.
            melhoria $\leftarrow$ true
            *break*
        **end if**
        $j \leftarrow j + 1$
    **end while**
    $i \leftarrow i + 1$
**end while**
FIM

Otimização e Busca - **busca local** Algoritmo da Subida de Encosta (*Hill Climbing*).

**Algorithm 2:** Pseudocódigo candidato.

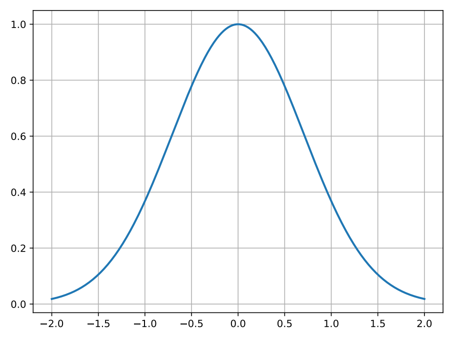
1: Definir o valor $\varepsilon$.
2: A partir de $x$ Gerar vizinho-candidato aleatório $y$ que respeite: $|x - y| <= \varepsilon$
3: FIM.

- Este algoritmo no que lhe concerne pode ser facilmente implementado em Python utilizando a biblioteca Numpy:
- **np.random.uniform**(low=x-E,high=x+E)

## Otimização e Busca - **busca local** Algoritmo da Subida de Encosta (*Hill Climbing*) Exemplo 1.

- Considere que deseja-se maximizar o problema contínuo:

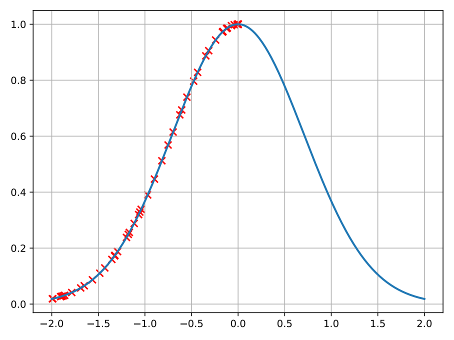
$$f(x) = e^{-x^2} \quad -2 \geq X \geq 2$$




- É um problema convexo (unimodal)?

- Considere que deseja-se maximizar o problema contínuo:

$$f(x) = e^{-x^2} \quad -2 \geq X \geq 2$$

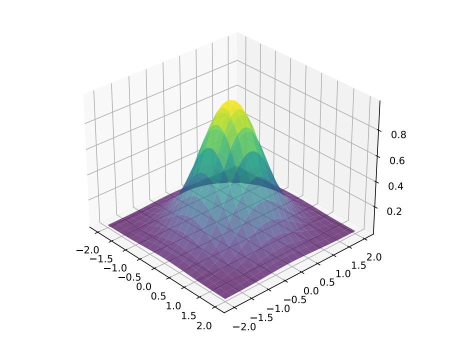



- É um problema convexo (unimodal)?

Otimização e Busca - **busca local** Algoritmo da Subida de Encosta (*Hill Climbing*) Exemplo 2.

- Considere que deseja-se maximizar o problema contínuo:

$$f(x,y) = e^{-x^2+y^2} \quad -2 \geq x \geq 2 \qquad -2 \geq y \geq 2$$

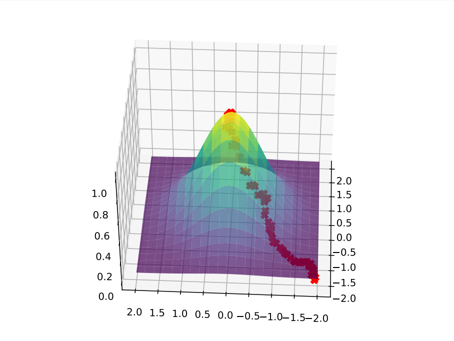



- É um problema convexo (unimodal)?

Otimização e Busca - **busca local** Algoritmo da Subida de Encosta (*Hill Climbing*) Exemplo 2.

- Considere que deseja-se maximizar o problema contínuo:

$$f(x,y) = e^{-x^2+y^2} \quad -2 \geq x \geq 2 \qquad -2 \geq y \geq 2$$

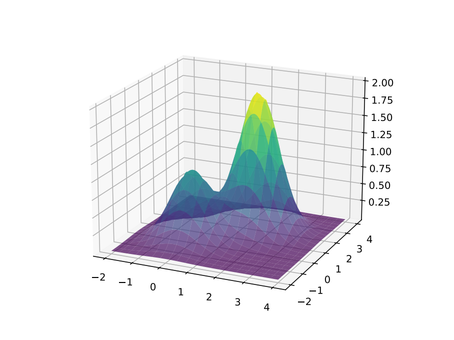



- É um problema convexo (unimodal)?

## Otimização e Busca - **busca local** Algoritmo da Subida de Encosta (*Hill Climbing*) Exemplo 2.

- Considere que deseja-se maximizar o problema contínuo:

$$f(x,y) = e^{-x^2+y^2} + 2e^{-((x-1.7)^2+(y-1.7)^2)} \qquad -2 \geq x \geq 4 \qquad -2 \geq y \geq 4$$

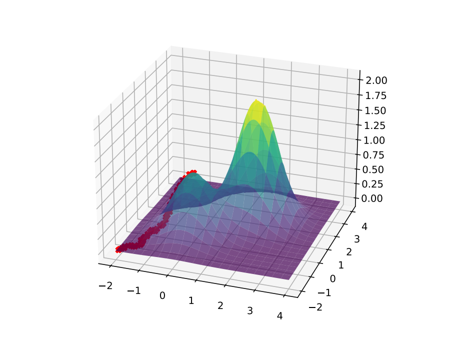



- É um problema convexo (unimodal)?

## Otimização e Busca - **busca local** Algoritmo da Subida de Encosta (*Hill Climbing*) Exemplo 2.

- Considere que deseja-se maximizar o problema contínuo:

$$f(x,y) = e^{-x^2+y^2} + 2e^{-((x-1.7)^2+(y-1.7)^2)} \quad -2 \geq x \geq 4 \qquad -2 \geq y \geq 4$$




- É um problema convexo (unimodal)?

## Otimização e Busca - Busca Aleatória (*Random Search*)

- Considerando a definição de função objetivo/custo anterior: $f : \mathbb{R}^p \rightarrow \mathbb{R}$, que produz uma saída escalar $y \in \mathbb{R}$ e possui $p$ ($p \geq 1$ variáveis de entrada.
- Pode-se formalmente escrever:

$$y = f(\mathbf{x}) = f(x_1, x_2, \cdots, x_p),$$

  em que $\mathbf{x}$ é um vetor cujas componentes são variáveis $x_i$, $i = 1, \cdots, p$
- Nos três exemplos anteriores, respectivamente, tem-se $p = 1$, $p = 2$, $p = 2$ e são funções contínuas e de variação suave.
- As duas primeiras, são **convexas**, ou seja, possuem apenas **um** ponto extremo chamado de máximo local.
- A última é um exemplo de função não convexa, tendo em vista que há dois picos. Um de máximo local e outro máximo global.

## Otimização e Busca - Busca Aleatória (*Random Search*)

- Em um problema formal de otimização de problema contínuo, o valor da variável $x$ para o qual $y = f(x)$ produz seu maior valor, é chamado de valor **ótimo** de $x$ (ou $x_{opt}$).
- Nesse sentido, $x_{opt}$ também pode ser vinculado ao mínimo da função-custo.
- Se o problema envolve uma função com duas variáveis, **busca-se** um vetor **ótimo** $\mathbf{x}_{opt} = \begin{bmatrix} x_{opt} & y_{opt} \end{bmatrix}^T$.
- O problema de minimização de funções (sem restrição) pode ser formalizado matematicamente como: O vetor $\mathbf{x} \in \mathbb{R}^p$ é o vetor **ótimo** se:

$$f(\mathbf{x}_{opt}) < f(\mathbf{x}), \quad \forall \mathbf{x} \neq \mathbf{x}_{opt} \text{ OU } \mathbf{x}_{opt} = \arg\min_{\forall \mathbf{x}} f(\mathbf{x})$$

## Otimização e Busca - Busca Aleatória (*Random Search*)

- A vizinhança já discutida, também pode ser formalmente escrita na forma **restrição de caixa**.
- Essa é descrita na forma de intervalos com limites inferiores e superiores para cada uma das $p$ variáveis que compõem o **vetor solução** $\mathbf{x} \in \mathbb{R}^p$.
- Assim, escrevendo como a $j-$ésima componente de $\mathbf{x}$ como $x_j$, e seus limites inferior e superior como $x_j^l$ e $x_j^u$, pode-se escrever a restrição do tipo caixa como:

$$x_j^l \leq x_j \leq x_j^u$$

- Ou através da versão vetorial:

$$\mathbf{x}^l \leq \mathbf{x} \leq \mathbf{x}^u$$

- Nesta última, $\mathbf{x}^l$ e $\mathbf{x}^u$ contém respectivamente, os limites inferiores e superiores para todas as componentes do vetor solução $\mathbf{x}$.

## Otimização e Busca - Busca Aleatória (*Random Search*)

- **OBS:** O uso de "$\leq$"na última equação é uma *liberdade poética*, pois tal comparação não pode ser realizada entre dois vetores.
- Para lidar com valores que excedem o limite imposto pela restrição, podem-se utilizar os seguintes métodos:
  1. Gerar um número aleatório uniforme no intervalo $x_j^l \leq x_j \leq x_j^u$:

     $$x_j \sim U(x_j^l, x_j^u), \text{ se } x^j < x_j^l \text{ ou } x_j > x_j^u$$

     em que $u \sim U(a, b)$ representa um número aleatório uniformemente distribuído no intervalo (a,b).
  2. Forçar que a solução que ocasiona uma violação de limite, assuma o extremo limite mais próximo. Se $x^j < x_j^l$, então $x^j = x_j^l$. Se $x^j > x_j^u$, então $x^j = x_j^u$

## Otimização e Busca - Busca Aleatória Local (*Local Random Search*)

- O algoritmo de busca aleatória local (LRS, do inglês *local random search*), consiste em testar soluções candidatas em uma vizinhança próxima ao $\mathbf{x}_{best}$.
- É uma heurística estocástica, ao dependerem de rotinas que geram números aleatórios.
- Possui uma única solução a cada iteração.
- Não é um método bioinspirado.
- Não depende de gradiente
- Pode ser utilizado para funções descontínuas.

## Otimização e Busca - Busca Aleatória (*Local Random Search*)

- O algoritmo possui a seguinte sequência de passos:

1.1 Especificar as restrições do tipo caixa $\mathbf{x}^l \leq \mathbf{x} \leq \mathbf{x}^u$ para todas as variáveis (especificar o domínio).

1.2 Especificar o desvio-padrão $\sigma$ do ruído (perturbação aleatória) a ser adicionado à melhor solução corrente para gerar candidatos.

2 Inicializar a solução melhor inicial $\mathbf{x}_{best} \sim U(\mathbf{x}^l, \mathbf{x}^u)$.

3 Avaliar a melhor solução inicial $\mathbf{x}_{opt}$, ou seja, calcular $f(\mathbf{x}_{opt})$.

4 Gerar uma solução candidata $\mathbf{x}_{cand}$ com valor $\mathbf{x}_{best} + \mathbf{n}$. Em que $\mathbf{n} \in \mathbb{R}^d$ segue uma distribuição normal variada de vetor médio nulo e matriz covariância $\sigma^2\mathbf{I}_d$, ou seja, $\mathbf{n} \sim N(\mathbf{0}_d, \sigma^2\mathbf{I}_d)$. OBS: Caso haja violação das restrições, aplicar um dos métodos descritos anteriormente.

5 Avaliar a melhor solução candidata $\mathbf{x}_{cand}$, ou seja, calcular $f(\mathbf{x}_{cand})$.

## Otimização e Busca - Busca Aleatória (*Local Random Search*)

- O algoritmo possui a seguinte sequência de passos:
  6. Verificar como a solução candidata está com relação à melhor solução corrente.

$$Se \;\; f(\mathbf{x}_{cand}) > f(\mathbf{x}_{best}), \; \text{Então, } \mathbf{x}_{best} = \mathbf{x}_{cand} \quad f(\mathbf{x}_{best}) = f(\mathbf{x}_{cand})$$

  7. Vá para o passo 4 até o algoritmo não ter convergido ( permaneça inalterado por um certo número de iterações) ou até o número máximo de iterações ($N_{max}$) seja atingido.
  8. Se uma das condições do passo 7 seja verdadeira, encerre a execução do algoritmo e retorne $\mathbf{x}_{best}$ e $f(\mathbf{x}_{best})$

## Otimização e Busca - busca aleatória local (LRS).

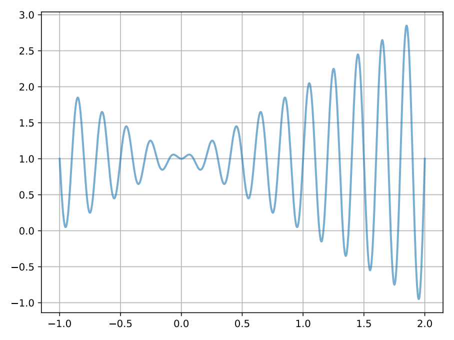
**Algorithm 3:** Pseudocódigo busca aleatória local.

Definir uma quantidade máxima de iterações $N_{max}$.<br>
Definir $\mathbf{x}^l$ e $\mathbf{x}^u$.<br>
Definir valor de $\sigma$ (perturbação aleatória).<br>
$\mathbf{x}_{best} \sim U(\mathbf{x}^l, \mathbf{x}^u)$<br>
$f_{best} = f(\mathbf{x}_{best})$<br>
$i \leftarrow 0$<br>
**while** $i < N_{max}$ **do**<br>
    $\mathbf{n} \leftarrow \sim N(0, \sigma)$<br>
    $\mathbf{x}_{cand} = \leftarrow \mathbf{x}_{best} + \mathbf{n}$<br>
    Verificar a violação da restrição em caixa.<br>
    $f_{cand} = f(\mathbf{x}_{cand})$<br>
    **if** $f_{cand} > f_{best}$ **then**<br>
        $\mathbf{x}_{best} = \mathbf{x}_{cand}$<br>
        $f_{best} = f_{cand})$<br>
    **end if**<br>
**end while**<br>
FIM.

## Otimização e Busca - **busca aleatória local** Exemplo.

- Considere que deseja-se maximizar o problema contínuo:

$$f(x) = x \cdot \sin(10 \cdot \pi \cdot x) + 1$$

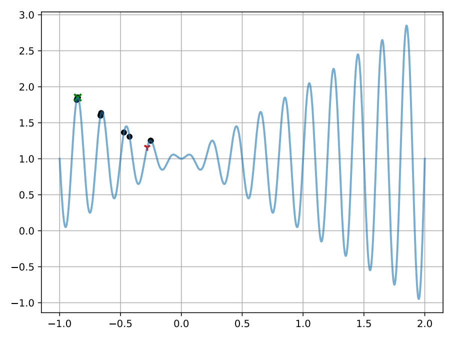



- É um problema convexo (unimodal)?

## Otimização e Busca - **busca aleatória local** Exemplo.

- Considere que deseja-se maximizar o problema contínuo:

$$f(x) = x \cdot \sin(10 \cdot \pi \cdot x) + 1$$

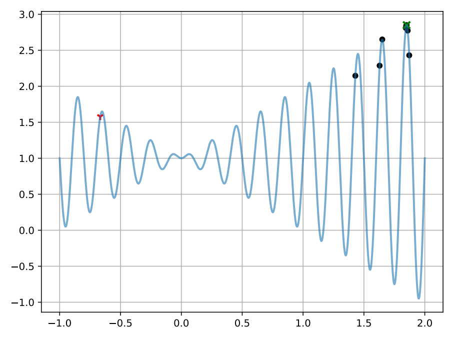



- É possível encontrar o máximo global (para este caso). Se sim, o que deve ser feito?

## Otimização e Busca - **busca aleatória local** Exemplo.

- Considere que deseja-se maximizar o problema contínuo:

$$f(x) = x \cdot \sin(10 \cdot \pi \cdot x) + 1$$




- É possível encontrar o máximo global (para este caso). Se sim, o que deve ser feito?

## Otimização e Busca - Busca Aleatória Global (*Random Search*)

- O algoritmo de busca aleatória global (GRS), consiste em testar soluções candidatas que são geradas aleatoriamente dentro do domínio da função a ser otimizada.
- Esse método será utilizado para encontrar o ponto ótimo e o valor ótimo correspondente da função de interesse.
- O algoritmo é baseado em heurística, pois, não é derivado a partir de princípios matemáticos formais, mas sim de conhecimento intuitivo ou informal sobre o domínio do problema.
- Para determinados casos, não se conhece os pontos mínimos e máximos de uma função custo/objetivo. Dessa maneira, é um algoritmo que não possui garantia de encontrar a solução ótima.
- Para contornar tal problema, várias execuções do algoritmo devem acontecer. A identificação da solução subótima é dada pela solução com a maior **frequência** (moda das soluções).

## Otimização e Busca - Busca Aleatória (*Random Search*)

- O algoritmo a ser apresentado é **estocástico** pois:
  1. Exige de uma solução inicial aleatória. Ou seja, a solução de partida, é gerada aleatoriamente.
  2. A geração de um candidato potencial a cada iteração é dependente de rotinas em que são gerados números aleatórios.
- É um método de solução única, em que apenas uma solução-candidata é gerada a cada iteração. Diferente de métodos populacionais como GAs
- Não é um método de inspiração biológica, pois sua formulação possui motivações puramente computacionais. Diferente de algoritmos bioinspirados.
- É um método livre de gradiente, ou seja, não requer cálculo de gradientes para determinar as direções de atualização de soluções. Por isso, **podem** ser utilizados para funções descontínuas (discretas).

## Otimização e Busca - Busca Aleatória (*Random Search*)

- O algoritmo a ser apresentado é **estocástico** pois:
  1. Exige de uma solução inicial aleatória. Ou seja, a solução de partida, é gerada aleatoriamente.
  2. A geração de um candidato potencial a cada iteração é dependente de rotinas em que são gerados números aleatórios.
- É um método de solução única, em que apenas uma solução-candidata é gerada a cada iteração. Diferente de métodos populacionais como GAs
- Não é um método de inspiração biológica, pois sua formulação possui motivações puramente computacionais. Diferente de algoritmos bioinspirados.
- É um método livre de gradiente, ou seja, não requer cálculo de gradientes para determinar as direções de atualização de soluções. Por isso, **podem** ser utilizados para funções descontínuas (discretas).

## Otimização e Busca - Busca Aleatória (*Random Search*)

- O algoritmo possui a seguinte sequência de passos:
  1. Especificar as restrições do tipo caixa $\mathbf{x}^l \leq \mathbf{x} \leq \mathbf{x}^u$ para todas as variáveis (especificar o domínio).
  2. Inicializar a solução melhor inicial $\mathbf{x}_{best} \sim U(\mathbf{x}^l, \mathbf{x}^u)$.
  3. Avaliar a melhor solução inicial $\mathbf{x}_{best}$, ou seja, calcular $f(\mathbf{x}_{best})$.
  4. Gerar uma solução candidata $\mathbf{x}_{cand}$ com valores aleatórios uniformes: $\mathbf{x}_{cand} \sim U(\mathbf{x}^l, \mathbf{x}^u)$
  5. Avaliar a melhor solução candidata $\mathbf{x}_{cand}$, ou seja, calcular $f(\mathbf{x}_{cand})$.
  6. Verificar como a solução candidata está com relação à melhor solução corrente.

$$Se \;\; f(\mathbf{x}_{cand}) > f(\mathbf{x}_{best}), \text{ Então, } \mathbf{x}_{best} = \mathbf{x}_{cand} \quad f(\mathbf{x}_{best}) = f(\mathbf{x}_{cand})$$

  7. Vá para o passo 4 até o algoritmo não ter convergido ( permaneça inalterado por um certo número de iterações) ou até o número máximo de iterações ($N_{max}$) seja atingido.
  8. Se uma das condições do passo 7 seja verdadeira, encerre a execução do algoritmo e retorne $\mathbf{x}_{best}$ e $f(\mathbf{x}_{best})$

## Otimização e Busca - busca aleatória global.

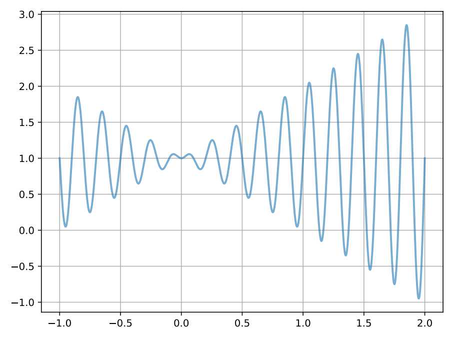
**Algorithm 4:** Pseudocódigo busca aleatória global.

Definir uma quantidade máxima de iterações $N_{max}$.<br>
Definir $\mathbf{x}^l$ e $\mathbf{x}^u$.<br>
$\mathbf{x}_{best} \sim U(\mathbf{x}^l, \mathbf{x}^u)$<br>
$f_{best} = f(\mathbf{x}_{best})$<br>
$i \leftarrow 0$<br>
**while** $i < N_{max}$ **do**<br>
    $\mathbf{x}_{cand} =\leftarrow\sim U(\mathbf{x}^l, \mathbf{x}^u)$<br>
    $f_{cand} = f(\mathbf{x}_{cand})$<br>
    **if** $f_{cand} > f_{best}$ **then**<br>
        $\mathbf{x}_{best} = \mathbf{x}_{cand}$<br>
        $f_{best} = f_{cand})$<br>
    **end if**<br>
**end while**<br>
FIM.

## Otimização e Busca - **busca aleatória global** Exemplo.

- Considere que deseja-se maximizar o problema contínuo:

$$f(x) = x \cdot \sin(10 \cdot \pi \cdot x) + 1$$

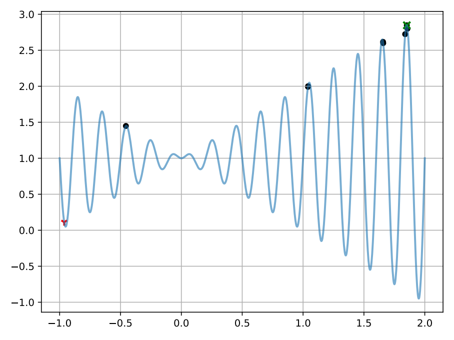



- É um problema convexo (unimodal)?

## Otimização e Busca - **busca aleatória local** Exemplo.

- Considere que deseja-se maximizar o problema contínuo:

$$f(x) = x \cdot \sin(10 \cdot \pi \cdot x) + 1$$




- É um problema convexo (unimodal)?

## Otimização e Busca - Têmpera Simulada (*Simulated annealing*)

- Algoritmo baseado em processo físico de têmpera.
- Tal procedimento é realizado para aumentar a dureza de certos metais ao aquecê-los em alta temperatura inicialmente e em seguida resfriando-os gradualmente.
- Na computação, é um algoritmo para solução de problema de otimização, através de uma técnica probabilística a fim de encontrar um ótimo global.
- Nesse sentido, pode ser utilizado para minimização da função-custo ou maximização da função-objetivo ($\max f(x)$ / $\min(-f(x))$).
- Imagine um exemplo em que será colocado uma bola de pingue-pongue em um dos gráficos não-convexos exibidos anteriormente.
- Deseja-se que essa bola atinja o ponto mínimo do gráfico.

## Otimização e Busca - Têmpera Simulada (*Simulated annealing*)

- Pode ser que ao soltar a bola, ela acabe em um mínimo local (a depender a da função-custo é muito provável que aconteça).
- Contudo, se agitarmos a superfície, a bola pode ser deslocada para fora do mínimo local.
- O truque está em agitar com força suficiente para fazer a bola sair dos mínimos locais, mas não o bastante para desalojá-la do mínimo global.
- A solução de têmpera simulada trata-se de agitar inicialmente com força (ou seja, em alta temperatura) e depois reduzir gradualmente a intensidade da agitação (baixar a temperatura).
- Diferente do algoritmo de subida de encosta, faz-se a escolha de um movimento aleatório. Caso essa escolha melhore a situação, essa é aceita. Caso contrário, o algoritmo aceita o movimento com alguma probabilidade menor que 1.

## Otimização e Busca - Têmpera Simulada (*Simulated annealing*)

- Pode ser que ao soltar a bola, ela acabe em um mínimo local (a depender a da função-custo é muito provável que aconteça).
- Contudo, se agitarmos a superfície, a bola pode ser deslocada para fora do mínimo local.
- O truque está em agitar com força suficiente para fazer a bola sair dos mínimos locais, mas não o bastante para desalojá-la do mínimo global.
- A solução de têmpera simulada trata-se de agitar inicialmente com força (ou seja, em alta temperatura) e depois reduzir gradualmente a intensidade da agitação (baixar a temperatura).
- Diferente do algoritmo de subida de encosta, faz-se a escolha de um movimento aleatório. Caso essa escolha melhore a situação, essa é aceita. Caso contrário, o algoritmo aceita o movimento com alguma probabilidade menor que 1.

## Otimização e Busca - Têmpera Simulada (*Simulated annealing*)

- Diferente do algoritmo de subida de encosta, faz-se a escolha de um movimento aleatório. Caso essa escolha melhore a situação, essa é aceita. Caso contrário, o algoritmo aceita o movimento com alguma probabilidade menor que 1.
- A probabilidade diminui exponencialmente com a "má qualidade"do vizinho. Esta também diminui à medida que temperatura é reduzida.
- Ou seja, utiliza-se uma abordagem de escalonamento da temperatura ao longo das iterações do algoritmo.
- Comumente, a aceitação probabilística é baseada na distribuição de Boltzmann-Gibbs dada por.

$$P_{ij} = e^{-\frac{f(x_j)-f(x_i)}{T}}$$

em que $x_j$ é o candidato, $x_i$ é o ótimo (corrente) e $T$ é a temperatura no instante atual.

## Otimização e Busca - busca aleatória global.

**Algorithm 5:** Pseudocódigo Têmpera Simulada.

Definir uma quantidade máxima de iterações $N_{max}$ e $T$.<br>
Definir $\mathbf{x}^l$ e $\mathbf{x}^u$.<br>
Definir valor de $\sigma$ (perturbação aleatória).<br>
$\mathbf{x}_{best} \sim U(\mathbf{x}^l, \mathbf{x}^u)$<br>
$f_{best} = f(\mathbf{x}_{best})$<br>
$i \leftarrow 0$<br>
**while** $i < N_{max}$ **do**<br>
    $\mathbf{n} \leftarrow\sim N(0, \sigma)$<br>
    $\mathbf{x}_{cand} =\leftarrow \mathbf{x}_{best} + \mathbf{n}$<br>
    Verificar a violação da restrição em caixa.<br>
    $f_{cand} = f(\mathbf{x}_{cand})$<br>
    **if** $f_{cand} < f_{best}$ **then**<br>
        $\mathbf{x}_{best} = \mathbf{x}_{cand}$<br>
        $f_{best} = f_{cand}$<br>
    **else if** $P_{ij} \geq U(0, 1)$ **then**<br>
        $\mathbf{x}_{best} = \mathbf{x}_{cand}$<br>
        $f_{best} = f_{cand}$<br>
    **end if**<br>
    $i \leftarrow i + 1$<br>
    escalona($T$)<br>
**end while**<br>
FIM.

## Otimização e Busca - Têmpera Simulada (*Simulated annealing*)

- Faz sentido destacar que a convergência do algoritmo pode mudar a partir do escalonamento definido.

$$T_{i+1} = 0.99 \cdot T_i$$

$$T_{i+1} = \frac{T_i}{1 + 0.99 \cdot \sqrt{T_i}}$$

$$T_{i+1} = \frac{T_i}{1 + \frac{T_0 - t_i}{(i-1)T_0 T_i} \cdot \sqrt{T_i}}$$

- Por exemplo, dada a função a seguir com domínio de $x \in \mathbb{R}, [-5, 5]$, construa uma implementação da têmpera simulada.

$$f(x) = -20e^{-0.2 \cdot |x|} - e^{\cos(2\pi \cdot x)} + 20 + e^1$$

## Otimização e Busca - Têmpera Simulada (*Simulated annealing*)

- Faz sentido destacar que a convergência do algoritmo pode mudar a partir do escalonamento definido.

$$T_{i+1} = 0.99 \cdot T_i$$

$$T_{i+1} = \frac{T_i}{1 + 0.99 \cdot \sqrt{T_i}}$$

$$T_{i+1} = \frac{T_i}{1 + \frac{T_0 - t_i}{(i-1)T_0 T_i} \cdot \sqrt{T_i}}$$

- Por exemplo, dada a função a seguir com domínio de $x \in \mathbb{R}$, $[-5, 5]$, construa uma implementação da têmpera simulada.

$$f(x) = -20e^{-0.2 \cdot |x|} - e^{\cos(2\pi \cdot x)} + 20 + e^1$$

## Otimização e Busca - Têmpera Simulada (*Simulated annealing*)

- Faz sentido destacar que a convergência do algoritmo pode mudar a partir do escalonamento definido.

$$T_{i+1} = 0.99 \cdot T_i$$

$$T_{i+1} = \frac{T_i}{1 + 0.99 \cdot \sqrt{T_i}}$$

$$T_{i+1} = \frac{T_i}{1 + \frac{T_0 - t_i}{(i-1)T_0 T_i} \cdot \sqrt{T_i}}$$

- Por exemplo, dada a função a seguir com domínio de $x_1 \in \mathbb{R}, [-1, 2]$ e $x_2 \in \mathbb{R}, [-1, 2]$ construa uma implementação da têmpera simulada.

$$f(x, y) = x^2 \sin(4 * \pi x) - y \sin(4 * \pi y + \pi) + 1$$