Além da busca clássica

Emílio Bergamim Júnior

Instituto de Geociências e Ciências Exatas - UNESP

2024

Conteúdo da aula

- Otimização e busca
- Subida da colina
- Recozimento simulado
- Busca local por feixes
- Algoritmo genético

Otimização e busca I

Na última aula, identificamos o problema de busca em espaços contínuos com o de resolução de equações. Isto é, resolver uma equação

$$\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0} \tag{1}$$

é análogo a fazer uma busca em um espaço contínuo e pode ser feito utilizando o método de Newton-Raphson.
Por vezes, o problema permite uma formulação ainda mais interessante na forma de um problema de otimização. Isto é, deseja-se encontrar $\boldsymbol{x}^* \in \mathbb{R}^n$ que satisfaça

$$\boldsymbol{x}^* = arg\max_{\boldsymbol{x}} f(\boldsymbol{x}) \quad \text{ou} \quad \boldsymbol{x}^* = arg\min_{\boldsymbol{x}} f(\boldsymbol{x}) \tag{2}$$

onde f é uma função com valores em $\mathbb{R}$. Esta é a forma geral de um **problema de otimização**.

Otimização e busca II

- A formulação enquanto um problema de otimização é útil pois o método de Newton-Raphson em altas dimensões demanda a inversão de uma matriz, o que possui complexidade cúbica.
- Como veremos a seguir, é possível uma forma simples de resolver problemas de otimização no caso em que f é diferenciável e possui um único extremo.
- Problemas de minimização podem ser transformados em problemas de maximização e vice-versa através da transformação

$$f(\boldsymbol{x}) \leftarrow -f(\boldsymbol{x}). \tag{3}$$

Logo, a solução de qualquer um destes implica em um método para resolver o problema de sinal oposto.

Descida (ou subida) do gradiente

Quando f é diferenciável, um algoritmo de otimização muito utilizado é a **descida do gradiente** (ou subida, no caso de maximização) que corresponde ao procedimento iterativo

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta \nabla f(\boldsymbol{x}_t) \tag{4}$$

para problemas de minimização e

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \eta \nabla f(\boldsymbol{x}_t) \tag{5}$$

para problemas de maximização.
O parâmetro $\eta > 0$ regula a **largura do passo** dado a cada iteração. A escolha de η é crucial para o funcionamento do algoritmo, já que um passo muito largo pode tornar a busca menos precisa, enquanto um passo muito curto pode levar a um algoritmo extremamente lento.

Descida (ou subida) do gradiente

O gradiente ∇f fornece a direção de maior crescimento da função. Isto significa que, seguindo na direção

$$\hat{\boldsymbol{r}} = \frac{\nabla f}{||\nabla f||}, \tag{6}$$

chega-se ao máximo de f, enquanto seguir na direção contrária do mesmo leva a um mínimo da função.
No entanto, como ∇f depende de $\boldsymbol{x}$, o algoritmo corresponde a uma série de aproximações locais de f por funções afim de forma a atingir o ponto crítico.

Critérios de parada

Em problemas contínuos, atingir o valor exato da função não é um objetivo realista, tanto pelas propriedades dos conjuntos contínuos como pela limitação de precisão numérica dos computadores.
Para esse tipo de algoritmo, pode-se usar como critérios de parada um número máximo de iterações t_{max} ou o critério de precisão do gradiente

$$||\nabla f(\boldsymbol{x}_t)|| < \epsilon \tag{7}$$

onde $\epsilon > 0$ é a precisão numérica desejada. Como os pontos críticos de funções diferenciáveis ocorrem em $\nabla f = 0$, este é um critério consistente para parada do algoritmo.

Um exemplo comportado: função quadrática

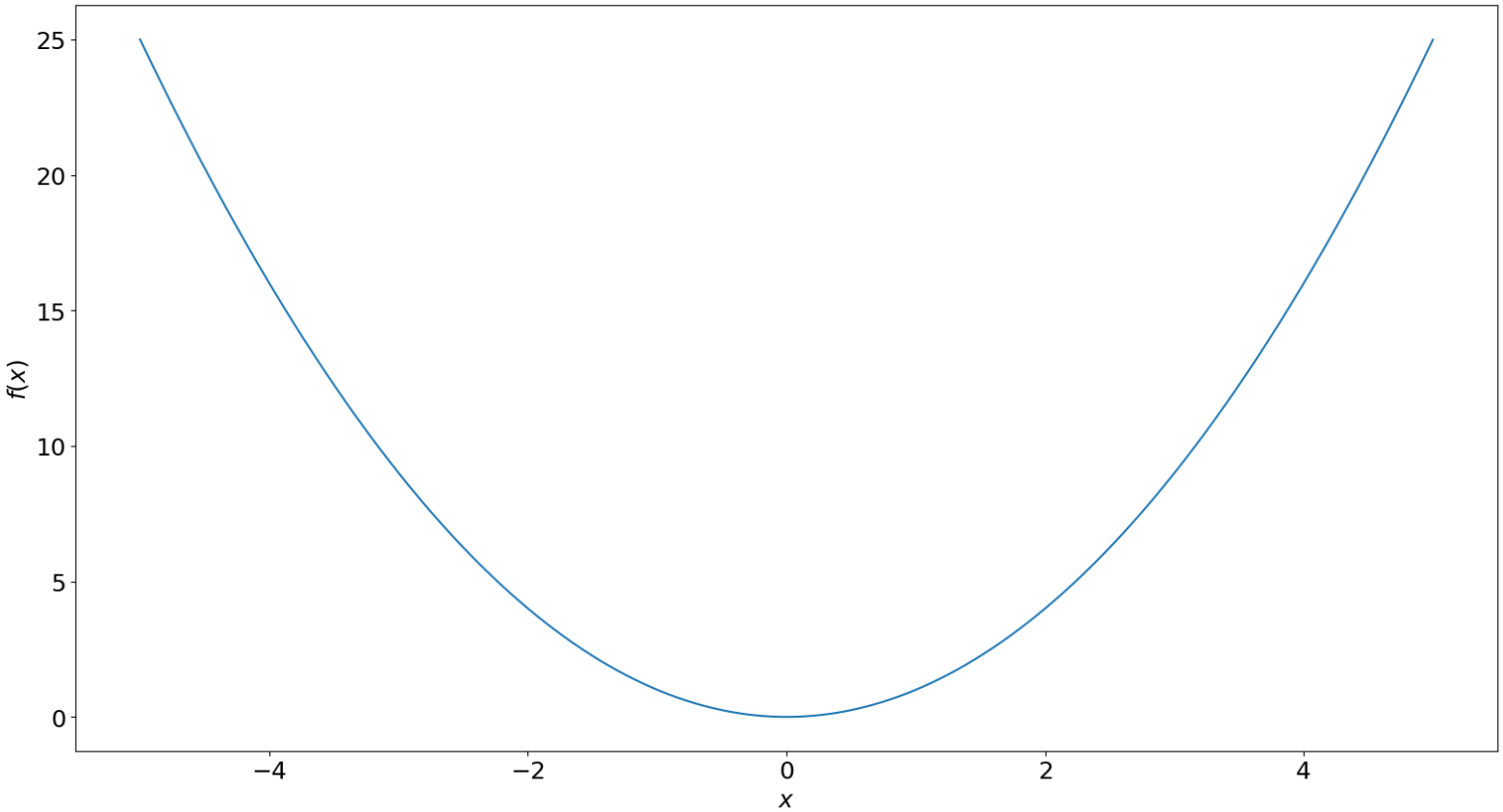

Figura: Gráfixo de $f(x) = x^2$.

Um exemplo comportado: função quadrática

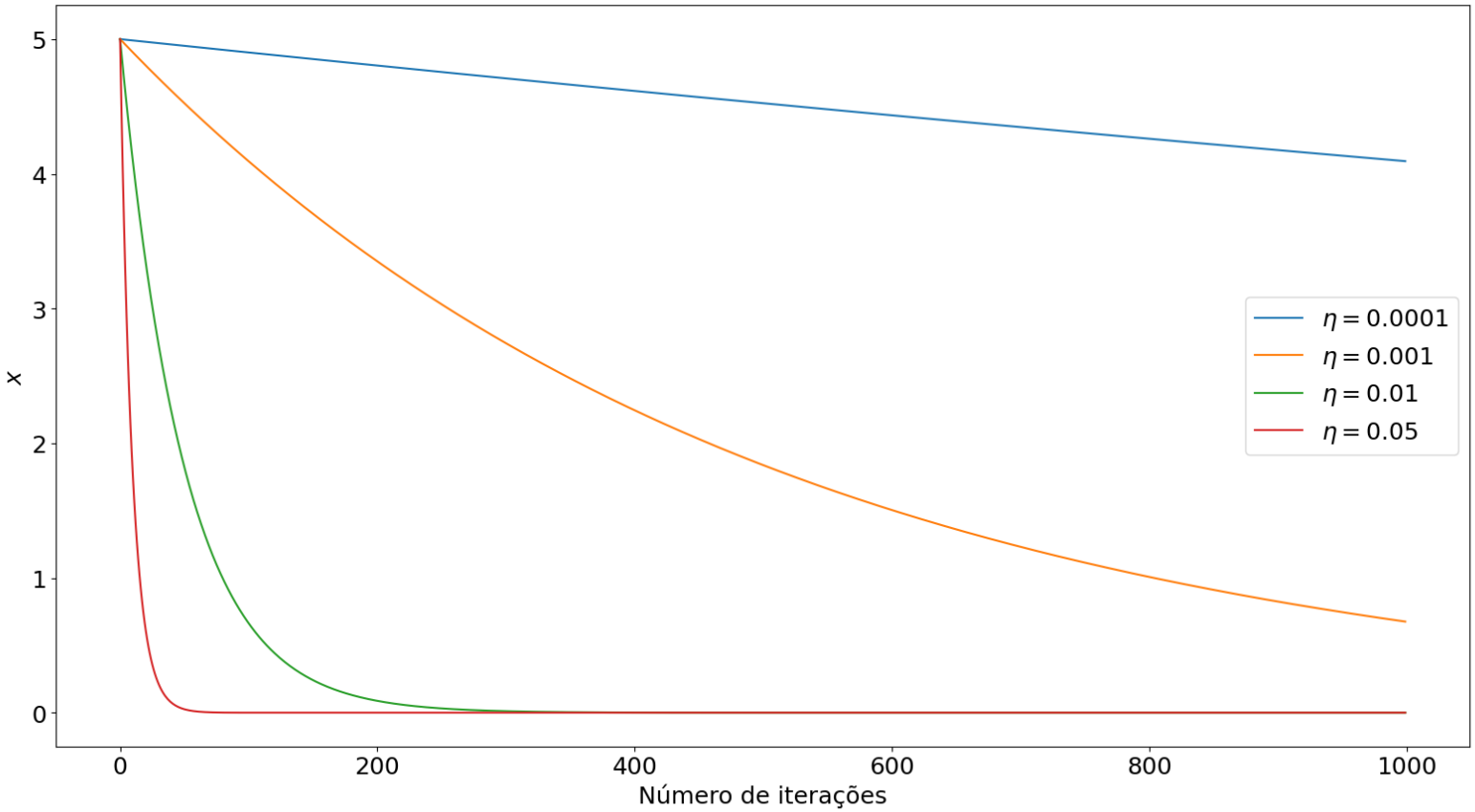

Figura: Evolução de x em função do número de iterações do algoritmo.

Um exemplo comportado: função quadrática

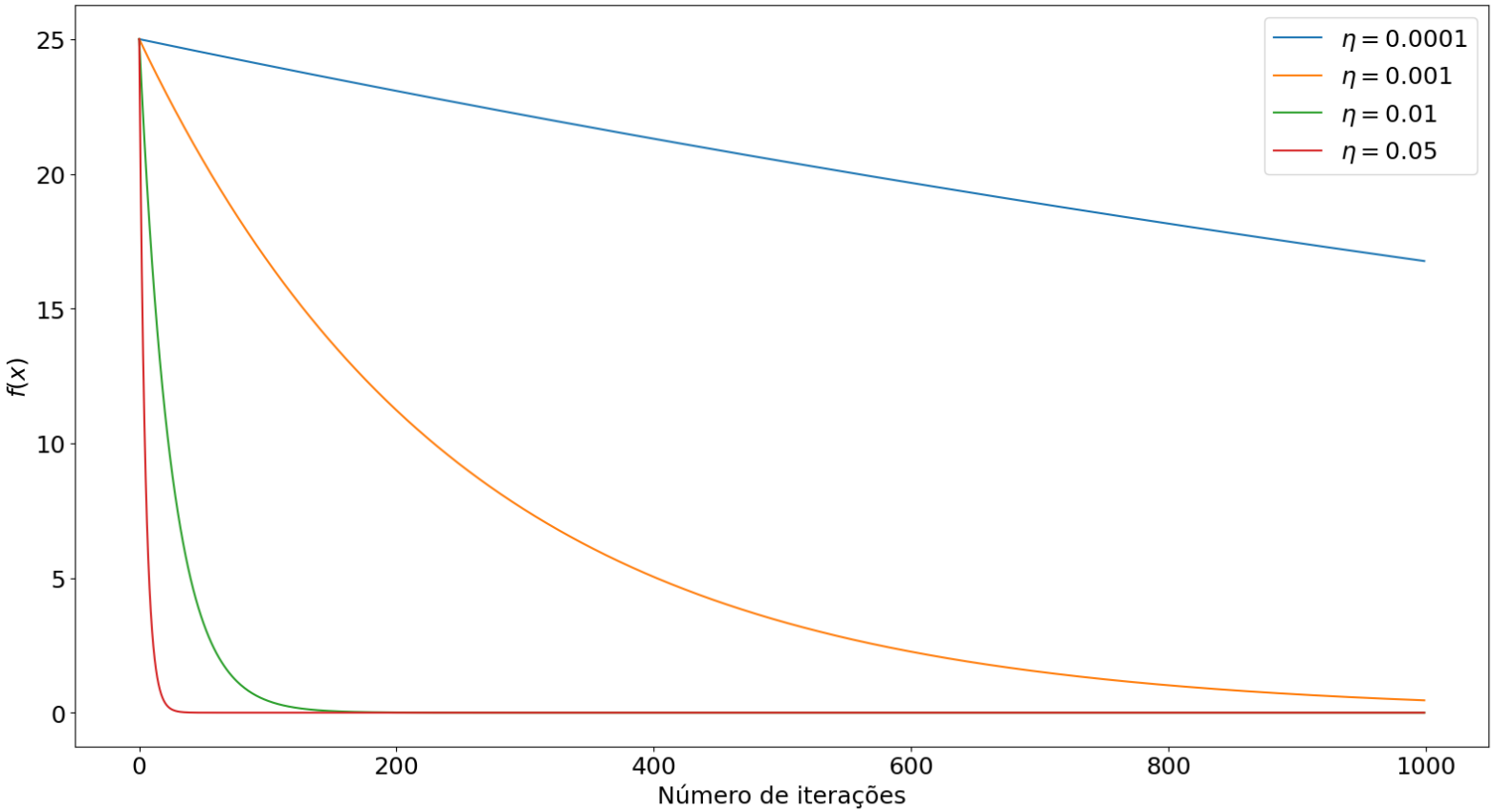

Figura: Evolução de $f(x)$ em função do número de iterações do algoritmo.

Um exemplo comportado: função quadrática

- Como o problema é unidimensional, o gradiente é igual à derivada:

$$\nabla f(x) = f'(x) = 2x. \tag{8}$$

- O parâmetro η regula a velocidade de convergência à solução.
- Neste caso, só existe um ponto no qual $\nabla f(x) = 0$. Logo, não há risco do algoritmo ficar preso em um mínimo local. Em verdade, a condição de que só existe um ponto crítico para a função é necessária para que o algoritmo opere desta forma independentemente da sua condição inicial.

Um exemplo patológico: função de Rastrigin

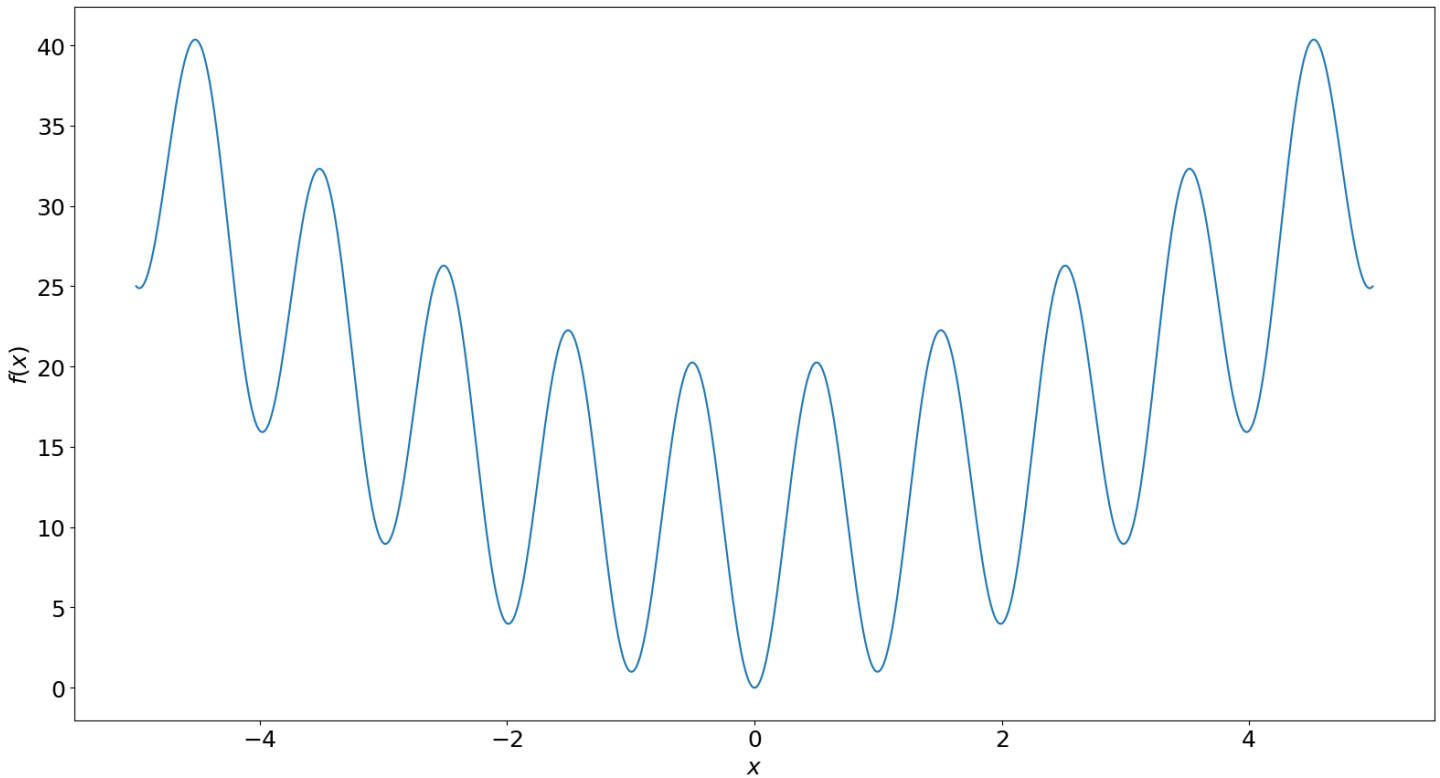

Figura: Gráfico da função de Rastrigin unidimensional $f(x) = 10 + x^2 - 10\cos(2\pi x)$.

Um exemplo patológico: função de Rastrigin

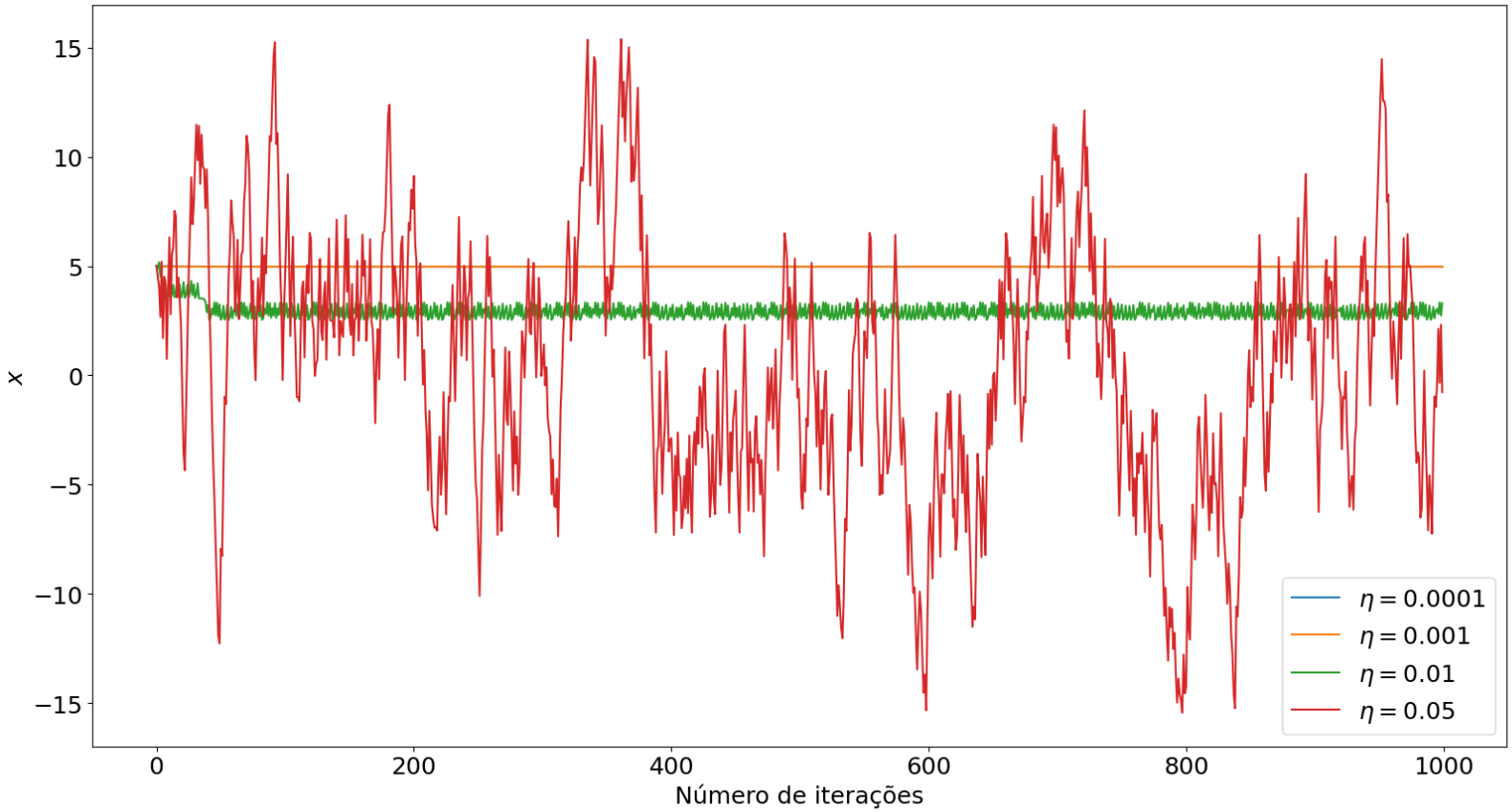

Figura: Evolução de x como função do número de iterações do algoritmo.

Um exemplo patológico: função de Rastrigin

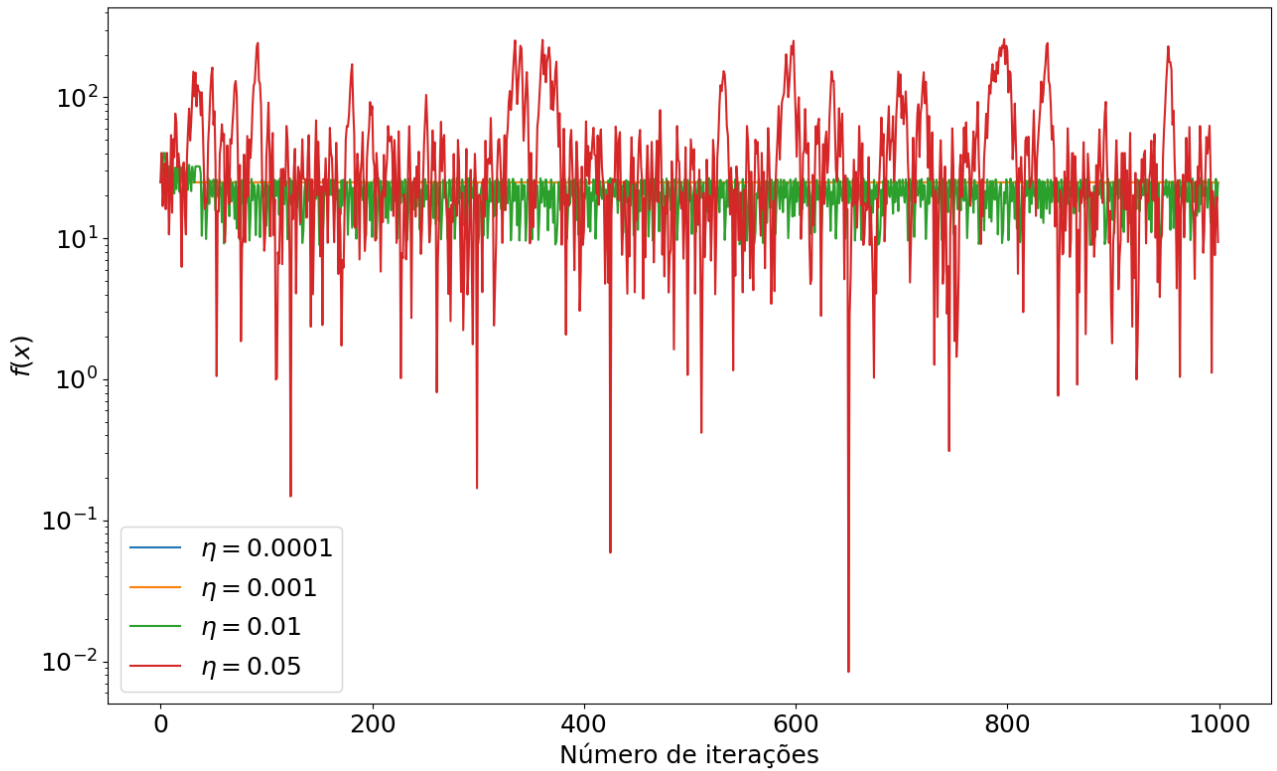

Figura: Evolução de $f(x)$ como função do número de iterações do algoritmo.

Um exemplo patológico: função de Rastrigin

- Novamente, por ser uma função unidimensional o gradiente é dado pela derivada $f'(x) = 2(x - 10\sin(2\pi x))$.
- No caso em que existem múltiplos pontos em que $\nabla f(x) = 0$, o algoritmo pode se prender a um mínimo local.
- Para η suficientemente pequeno, o algoritmo fica preso em um mínimo local.
- Para η suficientemente grande, o algoritmo oscila como um ruído, não se prendendo a nenhum mínimo local.
- Caso fosse escolhida uma condição inicial suficientemente próxima do mínimo global, o algoritmo poderia obter sucesso. Isso significa que a completude do algoritmo é **dependente da condição de inicialização**.

Outros problemas com a descida do gradiente

- A existência de múltiplos pontos críticos é um impedimento óbvio ao sucesso do algoritmo independente da condição inicial.
- O algoritmo também depende da possibilidade de calcular o gradiente de f para todos os pontos de seu domínio. No entanto, para determinadas funções o próprio cálculo do gradiente pode ser inviável, como é geralmente o caso de redes neurais.
- Em altas dimensões, a própria visualização da função torna-se impossível, de forma que a verificação da existência de mínimos locais é inviável.
- Para além disso, esse algoritmo só é funcional para funções contínuas e diferenciáveis, não se aplicando a problemas de otimização em domínios discretos.

Descida (subida) da colina I

- Assim como a busca gulosa, este algoritmo visa mover-se para um novo estado que esteja mais próximo da solução a partir do estado atual
- Sendo $G(x)$ uma função que gera um novo estado a partir de um estado x, a cada interação t faz-se

$$x_{prox} = G(x_t) \tag{9}$$

e x_{prox} é **aceito** caso, para a função objetivo $f(x)$ do problema seja satisfeito $f(x_{prox}) \leq f(x_t)$ e então faz-se

$$x_{t+1} = x_{prox} \tag{10}$$

e repete-se o procedimento.

- No caso de maximização, o critério de aceitação é invertido para $f(x_{prox}) \geq f(x_t)$

Descida (subida) da colina II

- O critério de parada do algoritmo deve envolver um número máximo de iterações t_{max} para evitar que o mesmo adentre um *loop* infinito.
- Para considerar que o algoritmo atingiu uma solução antes de atingir o máximo de iterações, pode-se verificar como se deu a variação dos último estados. Isto é, tomando uma quantidade t_{ult} das últimas iterações, vê-se se houve variação significativa nas iterações $\{x_{t-t_{ult}}, x_{t-t_{ult}+1}, x_{t-t_{ult}+2}, \ldots, x_t\}$.
- Nesse caso, pode-se investigar a variância de $f(x)$ nas últimas t_{ult} iterações e, caso esta seja inferior a uma tolerância $\epsilon > 0$, considera-se que uma solução foi encontrada.

A escolha de um novo estado I

- Em espaços discretos, $G(x_t)$ pode simplesmente retornar um estado x_{prox} que seja atingível a partir de x_t por meio de uma ação. No exemplo do *8-puzzle*, por exemplo, pode-se testar todos os movimentos possíveis e escolher aquele que minimiza uma heurística como o número de peças fora do lugar ou a distância de Manhattan.
  - Esse caso, no entanto, é de um espaço de estados relativamente pequeno. Caso exista uma quantidade muito grande de ações possíveis a partir de um estado x_t, pode-se adotar um esquema de sorteio, no qual sorteiam-se novos estados até que um destes seja aceito.
- Em espaços contínuos, um procedimento comum é a realização de uma translação aleatória: no caso unidimensional, por exemplo, sorteia-se a cada iteração um passo δ_t no intervalo $(-1, 1)$ e a função de atualização é

$$G(x_t) = x_t + \gamma\delta_t \tag{11}$$

e γ é a largura do passo.

A escolha de um novo estado II

- O intervalo no qual δ_t é gerado pode ser alterado a depender da função objetivo. Ou mesmo pode-se propor a geração de um passo a partir de outra distribuição de probabilidade, como uma gaussiana. Essas escolhas podem ser feitas de acordo com seu conhecimento prévio sobre a função objetivo ou pode-se experimentar diferentes formas de passo de forma a identificar qual leva aos resultados mais interessantes para sua aplicação.
- Em múltiplas dimensões, basta repetir o mesmo procedimento para cada uma das dimensões de x.

Exploração de um espaço de estados

- Como discutimos na última aula, uma sequência de melhores escolhas locais não necessariamente leva a uma solução do problema.
- O algoritmo de descida da colina padece do mesmo problema: uma vez que o mesmo só escolhe as melhores soluções a partir de um estado, depende-se que a estrutura do espaço de estados permita um caminho até a solução que constitua-se de escolhas localmente ótimas.
- Essa rigidez na escolha no próximo estado impede que o algoritmo explore o espaço de estados e conheça caminhos mais diversos que podem levar à solução

Escolhas aleatoriamente razoáveis

Sendo $f(x)$ a função objetivo do problema, considere dois estados sucessivos x_t e x_{prox}, sendo o segundo geraldo por algum dos procedimentos discutidos anteriormente.

- Em oposição à regra estrita de aceitação, pode-se utilizar uma relaxação desta:
  1. Se $f(x_{prox}) < f(x_t)$, o novo estado é aceito.
  2. Caso contrário, o mesmo é aceito com probabilidade

$$p = \min\{1, \exp(f(x_t) - f(x_{prox}))\}. \tag{12}$$

  3. Isso pode ser implementado sorteando uniformemente um valor r em $(0, 1)$ e aceitando o novo estado caso $r < p$ ou rejeitando-o em caso contrário.
- O algoritmo descrito acima é geralmente mais lento que a sua contraparte rígida, discutida anteriormente. Porém, seu objetivo é justamente garantir uma maior exploração do espaço de estados de forma a atingir soluções melhores. Este é comumente referido como **descida estocástica da colina**

Descida estocástica da colina

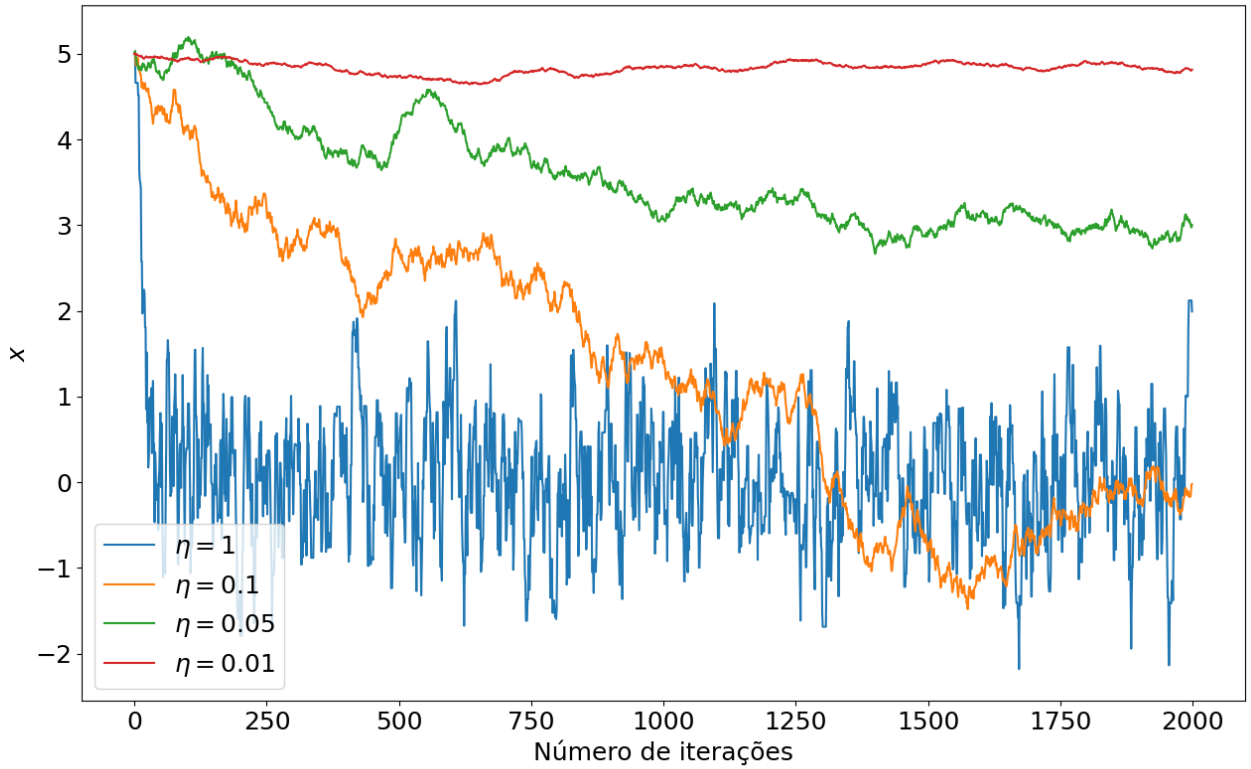

Figura: Evolução de x para a função quadrática utilizando o algoritmo de descida estocástica da colina.

Descida estocástica da colina

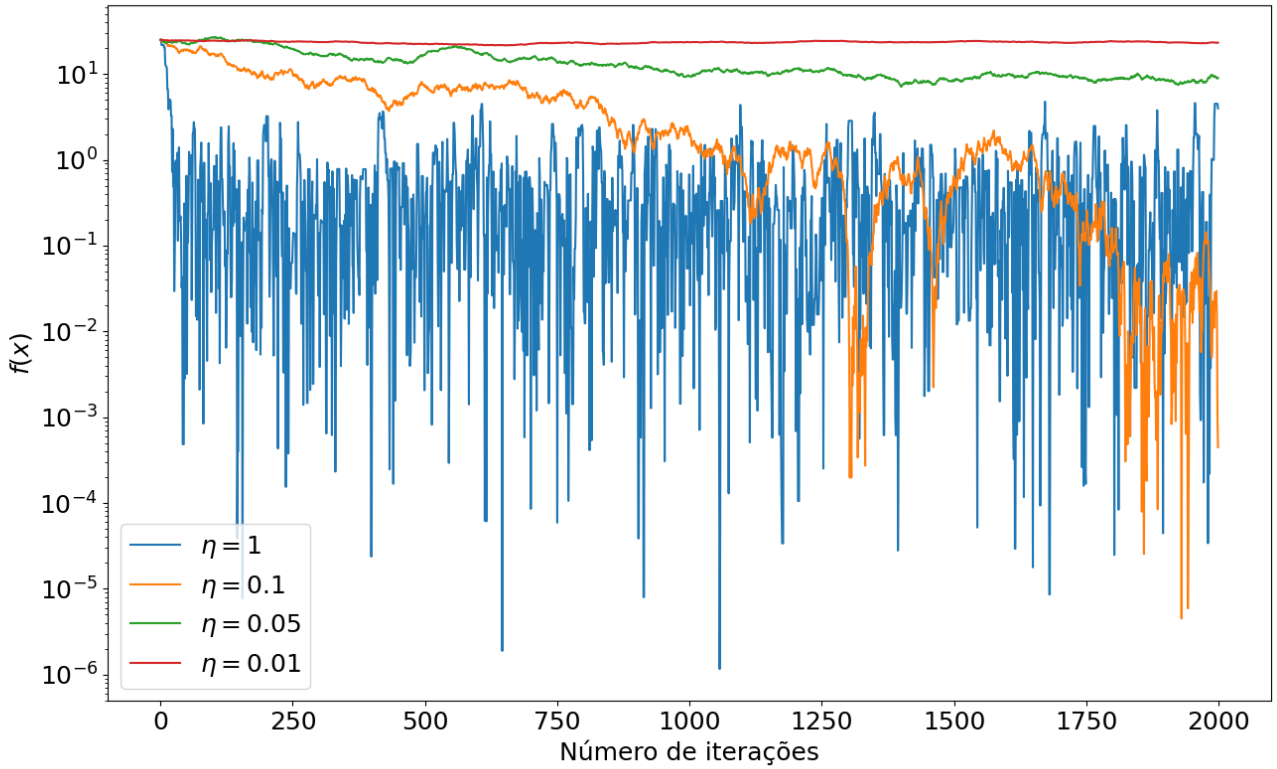

Figura: Evolução de $f(x)$ para a função quadrática utilizando o algoritmo de descida estocástica da colina.

Descida estocástica da colina

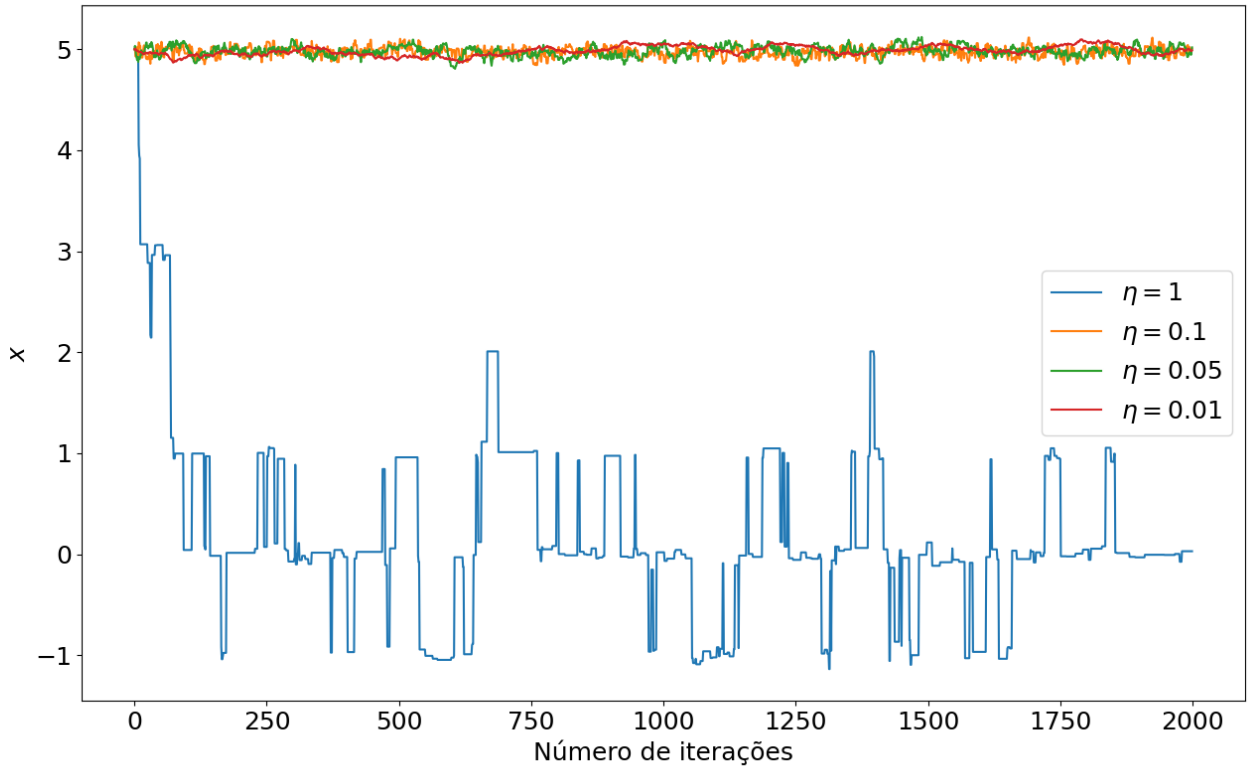

Figura: Evolução de x para a função de Rastrigin utilizando o algoritmo de descida estocástica da colina.

Descida estocástica da colina

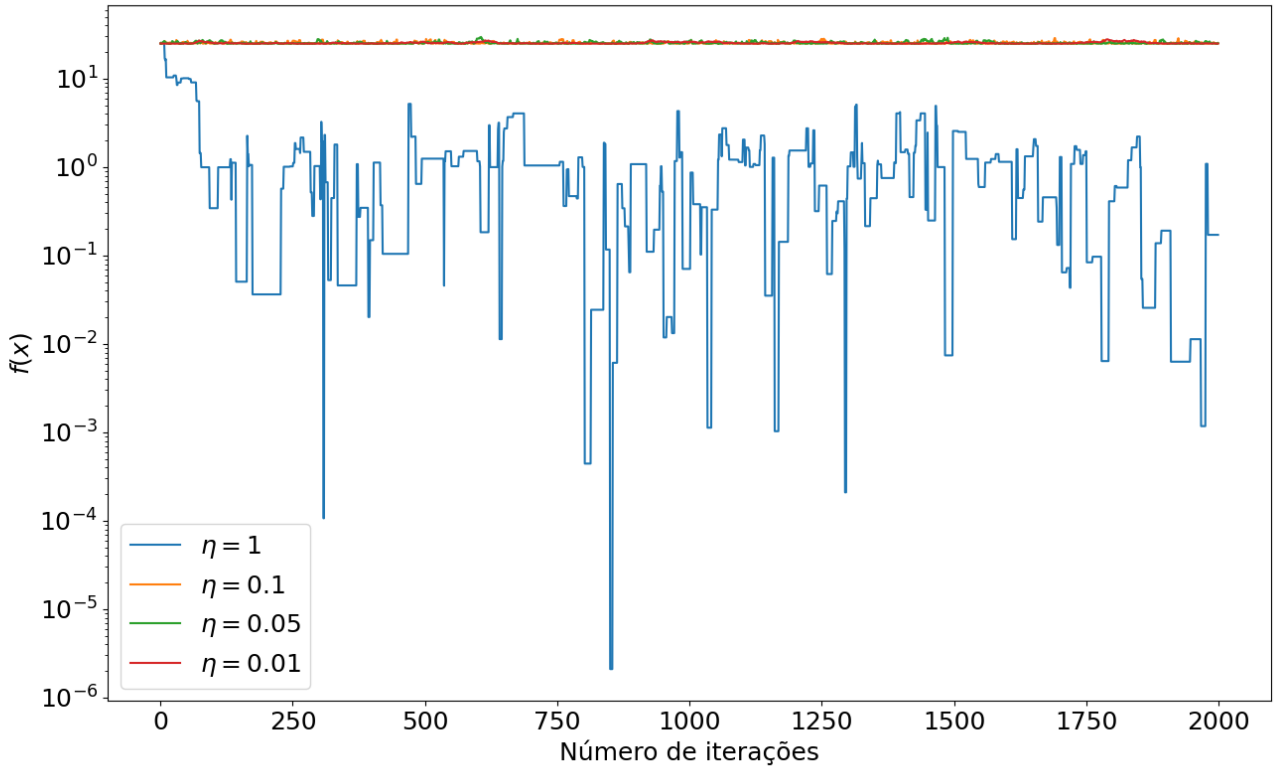

Figura: Evolução de $f(x)$ para a função de Rastrigin utilizando o algoritmo de descida estocástica da colina.

Recozimento simulado I

- O fator de aceitação p pode ainda depender de um parâmetro $\beta > 0$ na forma

$$p(x_{prox}; \beta) = \min\{1, \exp(\beta[f(x_t) - f(x_{prox})])\}. \tag{13}$$

- β é um parâmetro muito comum em uma disciplina da física conhecida como Mecânica Estatística e relaciona-se ao inverso da temperatura:

$$\beta = \frac{1}{k_B T} \tag{14}$$

sendo k_B a chamada constante de Boltzmann e $T > 0$ a temperatura absoluta do sistema.

- Em termos estatísticos, β regula a probabilidade de aceitar um estado x_{prox} tal que $f(x_{prox}) \geq f(x_t)$.

Recozimento simulado II

  - Conforme $\beta \to 0$, têm-se como consequência $p \to 1$, de forma que novos estados serão aceitos independentemente de diminuirem o valor de $f(x)$.
  - Em contrapartida, se $\beta \to \infty$, $p \to 0$, rejeitando todos os estados que não diminuem o valor de f.
- Esses casos extremos não são exatamente de interesse, mas ajudam a elucidar o que acontece entre os mesmos, que é onde pode-se atuar.
  - Para valores baixos de β, o algoritmo privilegia exploração em vez de otimização, garantindo que diversos estados sejam visitados.
  - Para valores maiores de β, a otimização é privilegiada em desfavor da exploração e pode ser compreendida como uma etapa de refinamento.
- Assim, começando em um valor pequeno de β, incrementa-se o mesmo a cada t_{inc} iterações de forma a começar o algoritmo explorando o espaço de estados e posteriormente restringindo as possibilidades de busca de forma a refinar o resultado da busca. Isso é o chamado **recozimento simulado**.

Recozimento simulado III

- O critério de parada nesse caso significa atingir um valor β_{max} definido previamente. Note que neste caso, como é possível escolher estados que aumentam o valor de f, é útil armazenar o melhor estado durante o algoritmo, que é aquele que será retornado ao final da execução.
- Portanto, existe um estado que é constantemente atualizado e um segundo que só é atualizado quando encontra-se um estado que resulta em um valor optimal de f dentre aqueles previamente observados.

Busca local por feixes I

- Uma outra forma de explorar o espaço de estados é realizar buscas paralelas e esporadicamente recomeçar a busca a partir do melhor estado obtido das buscas paralelas.
- Partindo de um estado x_t, é gerado um conjunto $S = \{x_{t+1,1}, x_{t+1,2}, x_{t+1,3}, ..., x_{t+1,m}\}$ oriundo de m ações possíveis a partir de x_t.
- Para cada um destes, o algoritmo de descida da colina é executado por t_{par} iterações.
- Ao fim dessas iterações, escolhe-se a melhor solução de acordo com o objetivo $f(x)$ e então repete-se o processo. O critério de parada é análogo ao discutido para a descida da colina. Isto é a chamada **busca local por feixes**.

Busca local por feixes II

- Pode-se ainda fazer a modificação estocástica sugerida para a descida da colina ao invés de usar o algoritmo mais rígido, de forma a obter melhores resultados de otimização. Isto é chamado **busca local por feixes estocástica**.
- Este tipo de algoritmo tem uma clara vantagem em relação aos discutidos anteriormente pois permite aproveitar-se de arquiteturas paralelas.
- Essa ideia pode ainda ser combinada com a ideia do recozimento simulado para produzir o algoritmo conhecido como **Parallel Tempering**. Neste caso, são conduzidas múltiplas execuções paralelas para diferentes valores de β e, após um número de iterações, faz-se uma redistribuição aleatória desses valores entre as configurações, de forma a intercalar otimização e exploração em execuções independentes.

Exercício sugerido

Qual a diferença entre fazer a busca local por feixes e reinicializar a configuração do algoritmo descida da colina a cada t_{par} iterações do segundo?

Algoritmo genético

- A busca estocástica por feixes locais descrita anteriormente pode ser compreendida como análoga a um processo de reprodução assexuada.
  - Após sucessivas interações da descida estocástica da colina, apenas o melhor elemento é utilizado para inicializar uma nova turma de estados.
- Uma alternativa é realizar a reprodução sexuada dos estados, na qual escolhem-se os melhores estados, que são posteriormente combinados (também chamado *crossover*) e então mutados. Essa é a essência do chamado **algoritmo genético**.
- Assim como o anterior, o algoritmo começa pela geração de um conjunto S de estados possíveis a partir de um estado inicial x_t.

Algoritmo genético: combinação I

- Para cada elemento de S, também dita a **população** de estados, são escolhidos dois estados para reprodução com probabilidade

$$p_{rep}(x) = \frac{1}{Z} \exp(-f(x)), \tag{15}$$

onde

$$Z = \sum_{x \in S} \exp(-f(x)). \tag{16}$$

- Gerar número aleatórios a partir de uma distribuição discreta não-uniforme não é um problema dos mais simples. Você pode utilizar as ferramentas disponíveis na *GNU Scientific Library* (GSL), que estão descritas aqui na seção *General Discrete Distributions*.

Algoritmo genético: combinação II

- Esses dois estados são então combinados de forma a produzir um novo estado. Para estados contínuos, pode-se escolher uma média simples:

$$x_{comb} = \frac{x_A + x_B}{2} \tag{17}$$

  que é um exemplo dentre muitas regras de combinação possíveis.
- Já para estados discretos, a escolha de uma regra combinação depende da semântica de cada estado.
  - Por exemplo, no *8-puzzle*, qual seria uma regra para combinar dois estados do jogo?
- Uma regra comum (quando possível), é quebrar os vetores x_A e x_B em partes e concatená-las de forma a produzir x_{comb}.

Algoritmo genético: mutação

- Parte da reprodução sexuada é o processo de mutação, no qual um novo estado não é meramente a combinação dos estados, mas está sujeito a mutações que não podem ser descritas a partir dos estados originários.
- Para estados contínuos, isto pode ser feito a partir da adição de um ruído ζ, que pode ter distribuição uniforme em um intervalo ou gaussiana, como já discutido para outros algoritmos:

$$x_{prox} = x_{comb} + \zeta \tag{18}$$

- Em estados discretos, se x_{comb} pode ser compreendido como um vetor de inteiros, uma possibilidade é escolher algumas entradas do mesmo e alterá-las aleatoriamente.
- Esse procedimento é então repetido até um número t_{max} de iterações ou até que a variância de f nas t_{ult} últimas iterações seja inferior a uma tolerância $\epsilon > 0$.

Atividade

- Implemente o algoritmo de recozimento simulado.
- Implemente o algoritmo de busca local por feixes.
- Faça ambos para domínios contínuos.

Trabalho - Otimização discreta

Implemente os algoritmos abaixo para o *8-puzzle* utilizando como funções objetivo o número de peças fora do lugar e a distância de Manhattan:

1. Descida estocástica da colina
2. Recozimento simulado (utilize pelo menos 5 valores de β)
3. Busca estocástica por feixes locais (implemente o conjunto S com pelo menos 5 elementos)

- A cada passo, você pode escolher um deslocamento aleatório do espaço vazio no tabuleiro.
- Compare com a busca em largura e a busca A^* (utilizando as funções objetivo menciondas anteriormente como heurística) em termos de tempo de execução e número de movimentos no tabuleiro para chegar na solução.
- Trabalhe com pelo menos dez configurações iniciais distintas. Calcule média e desvio padrão dos seus resultados
- Apresente seus resultados na forma de tabelas ou gráficos.

Trabalho - Otimização discreta

- Os algoritmos vistos na última aula, como já discutido, carregam garantias de convergência, além de optimalidade, então não espere que seu algoritmo seja tão competitivo, mas tente fazer o seu melhor.
- Além disso, você pode inserir controles para evitar que os algoritmos desta aula rodem por um tempo muito grande.
- Vocês possuem liberdade para escolher os parâmetros pertinentes da forma que julgarem melhor, mas sempre visando obter melhores resultados de otimização.

Exercícios sugeridos

- Escreva um pseudocódigo para o algoritmo de recozimento simulado
- Escreva um pseudocódigo para o algoritmo genético
- Dica: consulte o livro por Russell e Norvig