# Optimização Numérica

Rui Mendes

11 de Maio de 2016

- Conjunto de váriáveis numéricas ($X \in \Re^n$)
- Função objectivo $f : \Re^n \rightarrow \Re$
- Pretende-se descobrir um valor de $X$ para o qual a função objectivo tenha o seu máximo ou mínimo
- $\{X \in \Re^n \mid \forall Y \in \Re^n \; f(X) \leq f(Y)\}$ se estivermos a falar de minimização
- Estamos sobretudo interessados em optimização não linear
- E em casos em que não tenhamos informação sobre o gradiente

1. Começa-se com um valor inicial $X \in \Re^n$
2. Este valor poderá ser gerado aleatóriamente
3. Escolhe-se um valor $Y$ na vizinhança de $X$
4. Se a função objectivo tiver um valor melhor para $Y$ então passa-se para $Y$
5. Repete-se este processo enquanto houverem melhoras

# Optimização Local: Vantagens

- Fácil de implementar
- Rápido

# Optimização Local: Desvantagens

- Qual é a desvantagem deste método?
- A estagnação.
- Também conhecida por convergência prematura.
- O que é a convergência prematura?
- É quando o algoritmo encrava num óptimo local.
- Isso acontece porque todos os pontos na vizinhança têm soluções de qualidade pior do que a actual.

# Optimização Local: Desvantagens

- Qual é a desvantagem deste método?
- A estagnação.
- Também conhecida por convergência prematura.
- O que é a convergência prematura?
- É quando o algoritmo encrava num óptimo local.
- Isso acontece porque todos os pontos na vizinhança têm soluções de qualidade pior do que a actual.

# Optimização Local: Desvantagens

- Qual é a desvantagem deste método?
- A estagnação.
- Também conhecida por convergência prematura.
- O que é a convergência prematura?
- É quando o algoritmo encrava num óptimo local.
- Isso acontece porque todos os pontos na vizinhança têm soluções de qualidade pior do que a actual.

# Optimização Local: Desvantagens

- Qual é a desvantagem deste método?
- A estagnação.
- Também conhecida por convergência prematura.
- O que é a convergência prematura?
- É quando o algoritmo encrava num óptimo local.
- Isso acontece porque todos os pontos na vizinhança têm soluções de qualidade pior do que a actual.

# Optimização Local: Desvantagens

- Qual é a desvantagem deste método?
- A estagnação.
- Também conhecida por convergência prematura.
- O que é a convergência prematura?
- É quando o algoritmo encrava num óptimo local.
- Isso acontece porque todos os pontos na vizinhança têm soluções de qualidade pior do que a actual.

# Optimização Local: Desvantagens

- Qual é a desvantagem deste método?
- A estagnação.
- Também conhecida por convergência prematura.
- O que é a convergência prematura?
- É quando o algoritmo encrava num óptimo local.
- Isso acontece porque todos os pontos na vizinhança têm soluções de qualidade pior do que a actual.

# Simulated Annealing

- Uma solução;
- Começa-se com uma temperatura elevada;
- Aceitam-se sempre soluções melhores;
- Aceitam-se soluções piores com uma probabilidade que desce com a temperatura;
- A temperatura desce com o tempo;
- Para cada temperatura fazem-se várias iterações;
- Convenciona-se chamar energia à qualidade da solução;
- Uma solução melhor tem menos energia.

# SA: Probabilidade de Transição

$E_t$ Energia no tempo actual;

$E_{t-1}$ Energia no tempo anterior;

T Temperatura.

$\Delta E$ $E_t - E_{t-1}$

Probabilidade de transição

$$\Delta E <= 0 \implies 1$$
$$\Delta E > 0 \implies e^{\frac{-\Delta E}{T}}$$

# SA: Algoritmo

1. Gerar aleatóriamente a solução inicial;
2. Calcular a sua energia;
3. Se ainda não se atingiu a temperatura mínima;
4. Para cada uma de N experiências;
    1. Perturbar localmente a solução (pode depender da temperatura);
    2. Calcular a energia da nova solução;
    3. Se a energia da nova solução é menor aceita-se;
    4. Se for mais alta aceita-se com probabilidade $e^{\frac{-\Delta E}{T}}$;
5. Desce-se a temperatura e volta-se ao ponto 3.

1. $T_k = a \cdot T_{k-1}$ (ou escrito de outra maneira $T_k = a^k \cdot T_0$) em que $0 < a < 1$;
2. $T_k = \frac{c}{\log k + d}$ em que d é normalmente 1.
3. O esquema 1 é o mais usado, mas não garante a solução óptima.
4. O esquema 2 com um $c$ suficientemente elevado (a maior energia possível encontrada no problema) garante a convergência; mas é muito lento.

# SA: Parâmetros

- Temperatura inicial $T_0$
- Temperatura final ou nº de descidas de temperatura
- Nº de iterações para cada temperatura
- Esquema de arrefecimento (valor do $a$ para o esquema 1)
- Tipo de perturbação local à solução actual (e.g., mutação gaussiana)
- Se usar mutação gaussiana: $\sigma$ (desvio padrão da mutação)

- Procura local;
- Ao aceitar soluções de pior qualidade pode-se sair de mínimos locais de energia;
- À medida que a temperatura vai descendo, a probabilidade de aceitar soluções piores baixa;
- Se o arrefecimento for demasiado rápido o algoritmo estagna;
- Se o arrefecimento for demasiado lento o algoritmo não é eficiente;
- Teoricamente encontra sempre a solução óptima (se o arrefecimento for suficientemente lento);
- Pode funcionar em problemas de optimização combinatória.

# Differential Evolution

- Usa uma população de soluções;
- Só se aceitam soluções melhores;
- Há dois operadores:
  - Um esquema de mutação diferencial;
  - Cruzamento uniforme
- O cruzamento é entre o individuo actual e um gerado pelo esquema de mutação;
- O cruzamento só gera uma solução e não duas como nos algoritmos genéticos.

# DE: Esquemas de mutação

| | |
|---|---|
| DE/rand/1 | $x_1 + F \cdot (x_2 - x_3)$ |
| DE/best/1 | $x_b + F \cdot (x_1 - x_2)$ |
| DE/rand-to-best/1 | $x_1 + F \cdot (x_b - x_1) + F \cdot (x_2 - x_3)$ |
| DE/rand/2 | $x_1 + F \cdot (x_2 - x_3) + F \cdot (x_4 - x_5)$ |
| DE/rand/2 | $x_b + F \cdot (x_1 - x_2) + F \cdot (x_3 - x_4)$ |

- $x_i$ é a solução actual;
- $x_1, x_2, x_3, x_4, x_5$ são soluções todas diferentes entre si e diferentes de $x_i$;
- $x_b$ é a melhor solução da população
- $F$ é um número real (normalmente entre 0 e 1)

# DE: Geração de Nova Solução

1. Para cada $x_i$
2. Cria-se um $x_m$ usando o esquema de mutação (e.g., $x_m = x_1 + F \cdot (x_2 - x_3)$)
3. Cria-se $x_t$ através do cruzamento uniforme entre $x_i$ e $x_m$ com probabilidade CR (usa-se pelo menos um elemento de $x_m$)
4. Avalia-se a qualidade de $x_t$
5. $x_t$ substitui $x_i$ se tiver melhor qualidade
6. Caso contrário $x_i$ mantém-se na nova população

# DE: Algoritmo

1. Inicializa-se a população;
2. Avalia-se a população;
3. Para cada individuo da população $x_i$:
   1. Segue-se o procedimento do slide anterior para gerar $x_t$;
   2. Avalia-se a qualidade de $x_t$;
   3. $x_t$ substitui $x_i$ se tiver melhor qualidade
   4. Senão $x_i$ passa para a nova população
4. Passa-se a usar a nova população;
5. Salta-se para 3 até atingir o nº máximo de iterações.

# DE: Parâmetros

- Tamanho da população (normalmente é pequeno $\sim 20$);
- Esquema utilizado (normalmente DE/rand/1);
- F (com o esquema anterior 0.5 costuma funcionar bem);
- CR (com o esquema acima 0.6 costuma funcionar bem)

# DE: Discussão

- Muito rápido;
- Fácil de implementar;
- Poucos parâmetros;
- Muito robusto.

# Particle Swarm Optimization

- Inspirado em psicologia social e simulação de bandos de pássaros;
- Usa uma população de indivíduos;
- Cada indivíduo tem uma posição e uma velocidade;
- Em cada iteração, cada indivíduo actualiza a velocidade mediante uma aceleração usando dois componentes:
  - A atração em relação à melhor posição encontrada no passado;
  - A atração em relação à melhor posição encontrada pelo grupo no passado;

## Parâmetros

**Posição** Posição actual $\vec{x}_i$;

**Velocidade** Velocidade $\vec{v}_i$;

**Individualismo** A melhor posição alcançada $\vec{p}_i$;

**Conformismo** A melhor posição encontrada pelos seus vizinhos $\vec{p}_g$;

**Parâmetros** $\varphi_1 = \varphi_2 = 2.05, \chi = 0.729$.

## Algoritmo

$$\begin{cases} \vec{v}_i = & \chi(\vec{v}_i + \vec{\mathrm{U}}[0, \varphi_1](\vec{p}_i - \vec{x}_i) + \vec{\mathrm{U}}[0, \varphi_2](\vec{p}_g - \vec{x}_i)) \\ \vec{x}_i = & \vec{x}_i + \vec{v}_i \end{cases}$$

# PSO: Esquema geral

1. Inicializar a população;
2. Avaliar a qualidade dos indivíduos;
3. Para cada indivíduo:
   1. Escolher indivíduos na sua vizinhança;
   2. Imitar esses indivíduos;
   3. Actualizar a melhor experiência caso o valor actual seja melhor;
4. Iterar para 2 se ainda não atingimos o critério de paragem.

# PSO: Discussão

- Rápido;
- Contém elitismo;
- Fácil de implementar.

# Harmony Search

- Inspirado na improvisação da música Jazz;
- Usa um armazém de soluções (harmonias);
- Em cada iteração gera-se uma nova harmonia;
- A nova harmonia é gerada combinando partes das harmonias existentes no armazém;
- Existe uma componente de inspiração que modifica partes da nova harmonia;
- A harmonia gerada substitui a pior harmonia do armazém se for melhor do que esta.

# HS: Geração de Novas Harmonias

1. Para cada coordenada:
2. Com probabilidade **hmcr** (*Harmony Memory Considering Rate*):
    1. Usa-se o valor de uma coordenada duma harmonia escolhida aleatóriamente de dentro do armazém;
    2. Com probabilidade **par** (*Pitch Adjusting Rate*) perturba-se esse valor;
3. Se não se fez o ponto anterior escolhe-se um valor aleatório para essa coordenada;
4. Avalia-se a nova harmonia.

Se se pretender perturbar o valor de uma coordenada x:
$x = x + fw \cdot U[-1, 1] \cdot amplitude$
em que:
**fw** é o parâmetro *Fret Width*
**amplitude** é a amplitude dessa coordenada, i.e., se x pode ter valores no intervalo [a; b] então amplitude é b - a
U[-1, 1] dá valores aleatórios segundo a distribuição uniforme entre -1 e 1

# HS: Parâmetros

| | | |
|---|---|---|
| hms | Harmony Memory Size | $\sim 30$ |
| hmcr | Harmony Memory Considering Rate | $\sim 0.9$ |
| par | Pitch Adjusting Rate | $\sim 0.7$ |
| fw | Fret Width | $\sim [0.0001; 0.001]$ |

# HS: Discussão

- Simples de implementar;
- A selecção é interessante porque é a substituição da pior harmonia do armazém;
- A selecção é mais suave do que o DE;
- Pode ser adaptado a problemas de optimização combinatória

# Exercício

## Problemas

1. Esfera
2. Problema XOR
3. Problema Multiplexador

## Algoritmos

1. Simulated Annealing
2. Particle Swarm Optimization
3. Differential Evolution

Minimizar

$$
\begin{aligned}
f(x) &= \sum_{i=1}^{N} (x_i - a_i)^2 \\
N &= 30 \\
-100 \leq & \; x_i \leq 100 \\
-80 \leq & \; a_i \leq 80
\end{aligned}
$$

em que $a_i$ é escolhido previamente

Treinar uma feed forward neural network para o problema do XOR.

| $x_1$ | $x_2$ | r |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Usar uma rede com 2 neurónios na camada de entrada, 2 neurónios na camada intermédia e 1 neurónio de saída.

Treinar uma feed forward neural network para o problema do multiplexador.

| $x_1$ | $x_2$ | c | r |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 |

Usar uma rede com 3 neurónios na camada de entrada, 6 neurónios na camada intermédia e 1 neurónio de saída.