ALGORITMOS GENÉTICOS EM R

Mestrado Integrado em Engenharia Informática
Mestrado em Engenharia Informática
Computação Natural

João Ramos
Fábio Silva
Paulo Novais

Introdução

- Os algoritmos genéticos fazem parte da computação evolutiva, um campo da inteligência artificial.
- Um algoritmo genético (AG) é uma técnica de procura utilizada na ciência da computação para encontrar soluções aproximadas para problemas de otimização e procura, fundamentada principalmente pelo americano John Henry Holland.
- São inspirados pela teoria de evolução de Darwin.

Introdução

- Os algoritmos genéticos diferem dos algoritmos tradicionais de otimização em basicamente quatro aspetos:
  - *baseiam-se numa codificação do conjunto das soluções possíveis, e não nos parâmetros da otimização em si;*
  - *os resultados são apresentados como uma população de soluções e não como uma solução única;*
  - *não necessitam de nenhum conhecimento derivado do problema, apenas de uma forma de avaliação do resultado;*
  - *usam transições probabilísticas e não regras determinísticas*

Introdução

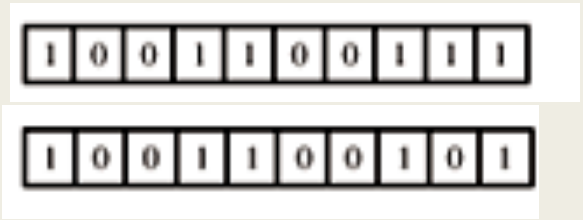
- Algumas das aplicações de algoritmos genéticos podem encontrar-se em:
  - *Otimização*
  - *Sistemas de controlo dinâmicos*
  - *Treino de redes neuronais*
  - *Descoberta de novas topologias conexionistas*
  - *Criatividade artificial*
  - *Problemas de planeamento de veículos*

Teoria

- Os algoritmos genéticos são uma classe particular de algoritmos evolutivos que usam técnicas inspiradas pela biologia evolutiva como:
  - *hereditariedade,*
  - *seleção natural*
  - *recombinação (ou crossing over);*
  - *mutação.*

Teoria

- Codificação de cromossomas
  - *O cromossoma deve de alguma maneira conter informação sobre a solução que ele representa. A forma mais utilizada para a codificação de cromossomas é sobre a forma de strings binárias.*

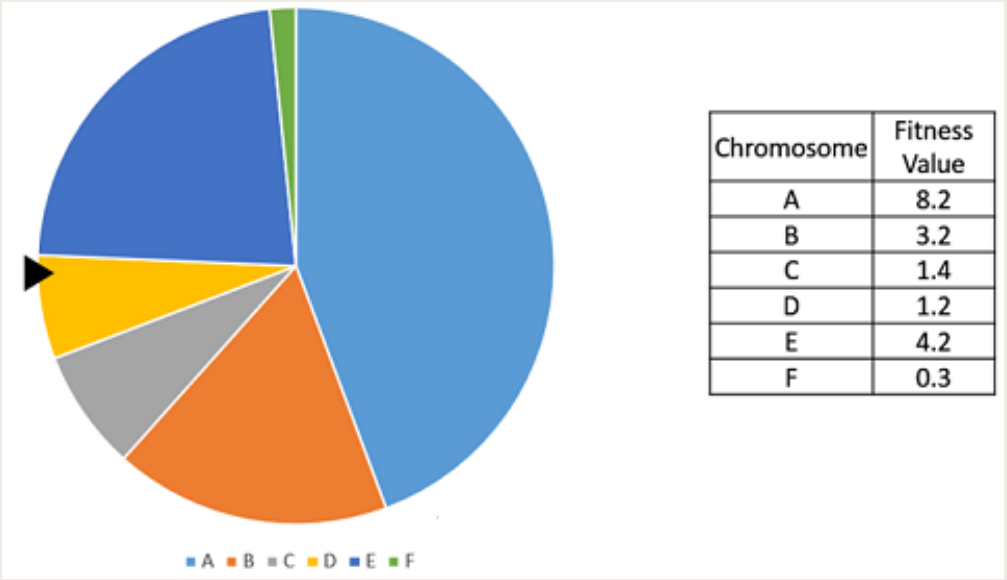


Teoria

- **Função objetivo ou fitness**
  - *A **função-objetivo** é usada para identificar os indivíduos (cromossomas) mais aptos. A grande vantagem dos algoritmos genéticos esta no fato de não precisarmos saber como funciona esta função objetivo, apenas tê-la disponível para ser aplicada aos indivíduos (cromossomas) e comparar os resultados.*

Teoria

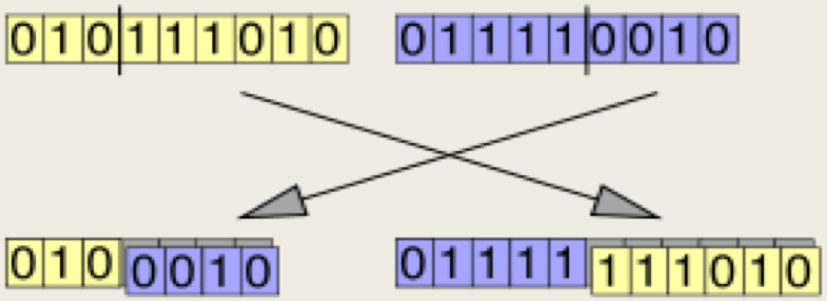
- **Seleção**
  - *Em geral, usa-se o algoritmo de seleção probabilística, onde os cromossomas são ordenados de acordo com a função-objetivo (fitness) e lhes são atribuídas probabilidades decrescentes de serem escolhidos - probabilidades essas proporcionais à razão entre a adequação do indivíduo e a soma das adequações de todos os indivíduos da população.*



Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

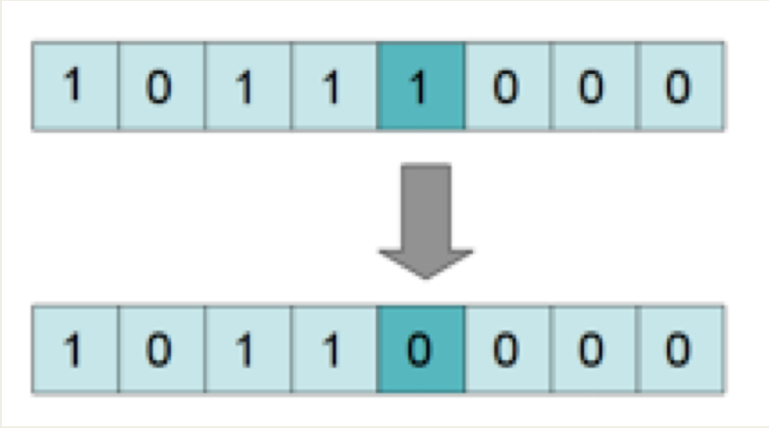
Teoria

- **Recombinação ou Crossover**
  - *Uma vez selecionados os indivíduos, estes passam, com uma probabilidade pré-estabelecida, pelo processo de cruzamento (crossover), onde partes dos genes dos pais são combinadas para geração de filhos.*



Teoria

- **Mutação**
  - *A mutação opera sobre os indivíduos e efetua algum tipo de alteração na sua estrutura. A importância deste operador reside no fato de garante que diversas alternativas serão exploradas.*

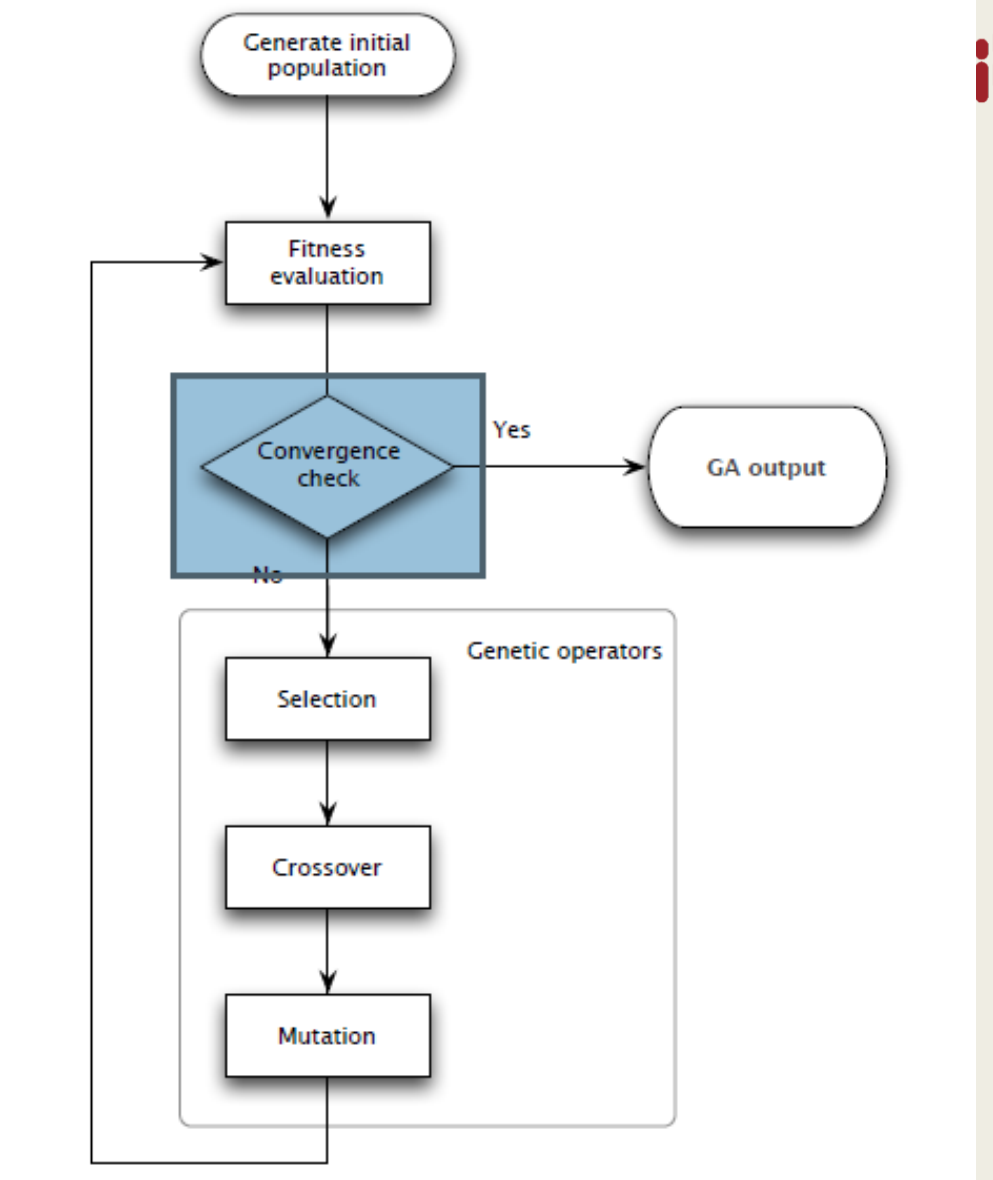


Teoria

Estrutura de um algoritmo genético básico

1. **[Início]** Gerar uma população aleatória de *n* cromossomas (apropriados para a solução do problema)
2. **[Fitness]** Avaliar cada cromossoma *x* na população através da função de fitness *f(x)*
3. **[Nova População]** Criar uma nova população através da repetição dos seguintes passos até a população estar completa
    - **[Seleção]** Escolher 2 cromossomas da população de acordo com as suas avaliações de fitness (quanto melhor o fitness melhor a sua probabilidade de ser selecionado)
    - **[Recombinação ou Crossover]** Tendo em conta a probabilidade de crossover, combinando 2 cromossomas formando assim um novo cromossoma.
    - **[Mutação]** Tendo em conta a probabilidade de mutação, mutar cromossomas na população).
    - **[Aceitação]** Por o novo cromossoma na nova população
4. **[Substituir]** Use os novos cromossomas criados na população de cromossomas para correr de novo o programa substituindo cromossomas da população anterior
5. **[Testar]** Se a condição de paragem é satisfeita, parar e devolver a melhor solução encontrada na população corrente.
6. **[Loop]** Ir para o passo 2

Teoria


Generate initial population
Fitness evaluation
Convergence check
Yes
GA output
No
Genetic operators
Selection
Crossover
Mutation

Teoria

- Parâmetros a utilizar:
  - *Tamanho da População*
    - O tamanho da população afeta o desempenho global e a eficiência dos AGs. Com uma população pequena o desempenho pode cair, pois deste modo a população fornece uma pequena cobertura do espaço de busca do problema. Uma grande população geralmente fornece uma cobertura representativa do domínio do problema, além de prevenir convergências prematuras para soluções locais ao invés de globais.
  - *Taxa de Cruzamento*
    - Quanto maior for esta taxa, mais rapidamente novas estruturas serão introduzidas na população. Mas se esta for muito alta, pode ocorrer perda de estruturas de alta aptidão. Com um valor baixo, o algoritmo pode tornar-se muito lento.

Teoria

- Parâmetros a utilizar:
  - *Taxa de Mutação*
    - Uma baixa taxa de mutação previne que uma dada posição fique estagnada em um valor, além de possibilitar que se chegue em qualquer ponto do espaço de busca. Com uma taxa muito alta a busca se torna essencialmente aleatória.
  - *Intervalo de Geração*
    - Controla a porcentagem da população que será substituída durante a próxima geração. Com um valor alto, a maior parte da população será substituída, mas com valores muito altos pode ocorrer perda de estruturas de alta aptidão. Com um valor baixo, o algoritmo pode tornar-se muito lento.

Algoritmos Genéticos em R

- Existem diversas alternativas para o uso de algoritmos genéticos em R.
- Packages para R:
  - *GA*
    - https://cran.r-project.org/web/packages/GA/GA.pdf
  - *Genalg*
    - https://cran.r-project.org/web/packages/genalg/genalg.pdf

Genalg

Genalg

- Instalação do package genalg no R

```
install.packages("genalg")
```

- Carregar o package GA

```
library(genalg)
```

Genalg

- Exemplo de representação de um cromossoma

```
cromossoma = c(1, 0, 0, 1, 1, 0, 0)
```

- Exemplo de uma função de fitness

```
evalFunc <- function(x) {
  return sum(x)
}
```

Genalg

- Função para correr o algoritmo genético

```
# run the model
iter = 100
GAmodel <- rbga.bin(size = 7, popSize = 200, iters = iter, mutationChance = 0.01,
                    elitism = T, evalFunc = evalFunc)
cat(summary(GAmodel))
plot(GAmodel)
```

Genalg

- Seleção do melhor resultado e teste da solução

```
bestSolution<-GAmodel$population[which.min(GAmodel$evaluations),]
dataset[bestSolution == 1, ]
#test solutions
solution = c(1, 1, 1, 1, 1, 0, 1)
dataset[solution == 1, ]
```

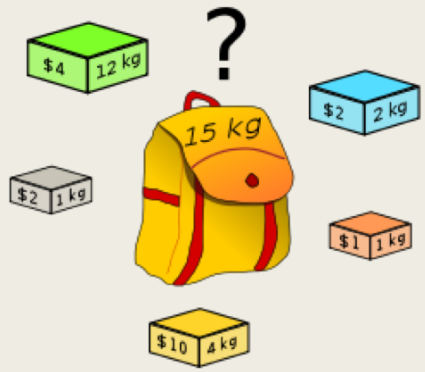
- Visualização de resultados

```
plot(GAmodel)
```

Genalg
Exercício (Knapsack Problem)

- Tenha em consideração o seguinte problema

  - *Uma pessoa vai estar 1 mês na natureza selvagem.*
  - *Só pode levar uma mochila com o peso máximo de 20 kg.*
  - *Poderá escolher um conjunto de itens, onde cada um tem um conjunto de pontos de sobrevivência.*
  - *Objetivo: otimizar o número de pontos de sobrevivência.*



Genalg
Exercício (Knapsack Problem)

- Tenha em consideração a seguinte tabela

Objecto	Pontos de Sobrevivência	Peso
Canivete	10	1
Feijões	20	5
Batatas	15	10
Saco de cama	2	1
Corda	30	7
Bússola	10	5
Uniões	30	1

Genalg
Exercício (Knapsack Problem)

- Inicialização da biblioteca genalg

```
install.packages("genalg")
install.packages("ggplot2")
install.packages("animation")

library(genalg)
library(ggplot2)
```

Genalg
Exercício (Knapsack Problem)

- Definição dos parâmetros em R

```
# DataSet
item <- c("pocketknife", "beans", "potatoes", "unions", "sleeping bag", "rope", "compass");
survivalpoints <- c(10, 20, 15, 2, 30, 10, 30);
weight <- c(1, 5, 10, 1, 7, 5, 1);
weightlimit <- 20;
dataset <- data.frame(item = item, survivalpoints = survivalpoints, weight = weight);

# Chromossome
chromosome <- c(1, 0, 0, 1, 1, 0, 0);
dataset[chromosome == 1, ]; # show active values for the chromosome
cat(chromosome %*% dataset$survivalpoints); #get the survival points
```

Genalg
Exercício (Knapsack Problem)

- Executar o algoritmo genético

```
evalFunc <- function(x) {
  current_solution_survivalpoints <- x %*% dataset$survivalpoints
  current_solution_weight <- x %*% dataset$weight

  if (current_solution_weight > weightlimit)
    return(0) else return(-current_solution_survivalpoints)
}

# run the model
iter = 100
GAmodel <- rbga.bin(size = 7, popSize = 200, iters = iter, mutationChance = 0.01,
                    elitism = TRUE, evalFunc = evalFunc)
```

Genalg
Exercício (Knapsack Problem)

- Verificar Resultados

```
cat(summary(GAmodel))
plot(GAmodel)

#copy the best solution
bestSolution<-GAmodel$population[which.min(GAmodel$evaluations),]
dataset[bestSolution == 1, ]
```

Genalg
Exercício (Knapsack Problem 2)

- Altere a capacidade da mochila no problema anterior bem como os pontos e pesos de cada elemento. Observe os diferentes resultados produzidos.

Genalg
Exercício (Knapsack Problem 3)

- Modifique o exercício anterior e adicione uma nova restrição baseada em pontos de alimentação assegurando que no plano existe sempre uma porção de comida.
- Para este efeito deve:
  - *Complementar dados sobre pontos de fome para cada objeto*
  - *Acrescentar novas opções ao problema*
  - *Alterar a função de fitness do problema*

GA

GA

- Instalação do package GA no R

```
install.packages("GA")
```

- Carregar o package GA

```
library(GA))
```

GA

- Representação de um cromossoma
  - *Permite o uso de valores reais assim como o uso de strings binárias*

- Exemplo de uma função de fitness
  - *No caso de se trabalhar com números reais pode-se utilizar diretamente a função a otimizar ou definir manualmente a função de otimização*

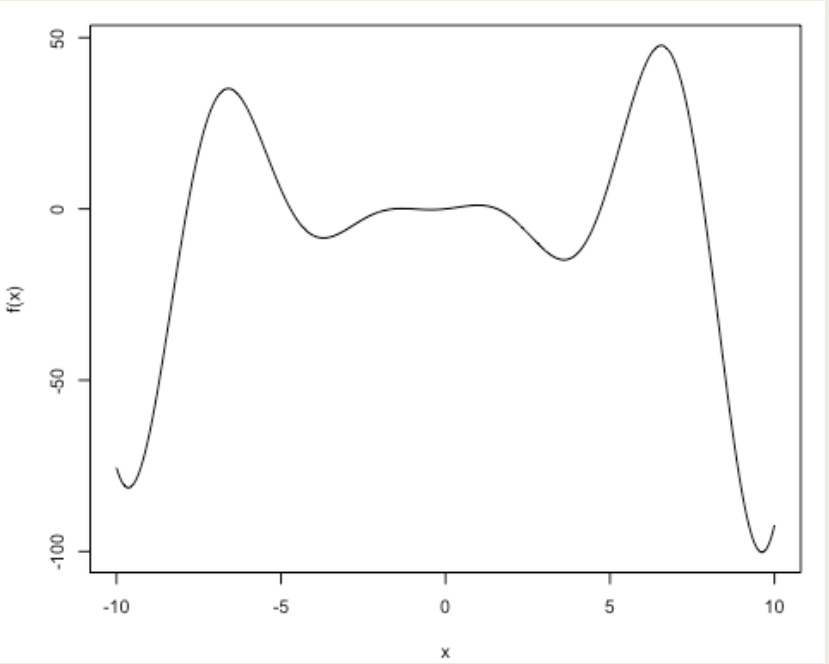
GA

- Exemplo da função para correr o algoritmo genético

```
GA <- ga(type = "real-valued", fitness = f, min = min, max = max,
        monitor = FALSE)
summary(GA)
```

GA – Exercício 1

- Encontrar o mínimo da função
  - *f <- function(x) (x^2+x)*cos(x)*



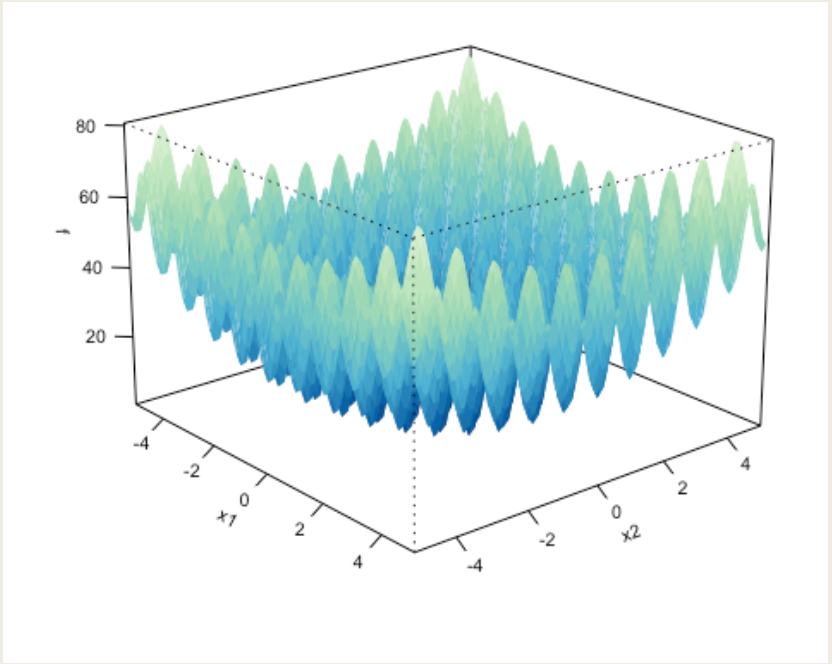
GA – Exercício 2

- Resolução

```
f <- function(x)  (x^2+x)*cos(x)
min <- -10; max <- 10
curve(f, min, max, n = 1000)

GA <- ga(type = "real-valued", fitness = f, min = min, max = max,
         monitor = FALSE)
summary(GA)

plot(GA)

#better looking graph
curve(f, min, max, n = 1000)
points(GA@solution, GA@fitnessValue, col = 2, pch = 19)
```

GA – Exercício 2

- Encontrar o máximo da função
  - *Rastrigin <- function(x1, x2) { 20 + x1^2 + x2^2 - 10*(**cos**(2*pi*x1) + **cos**(2*pi*x2)) }*



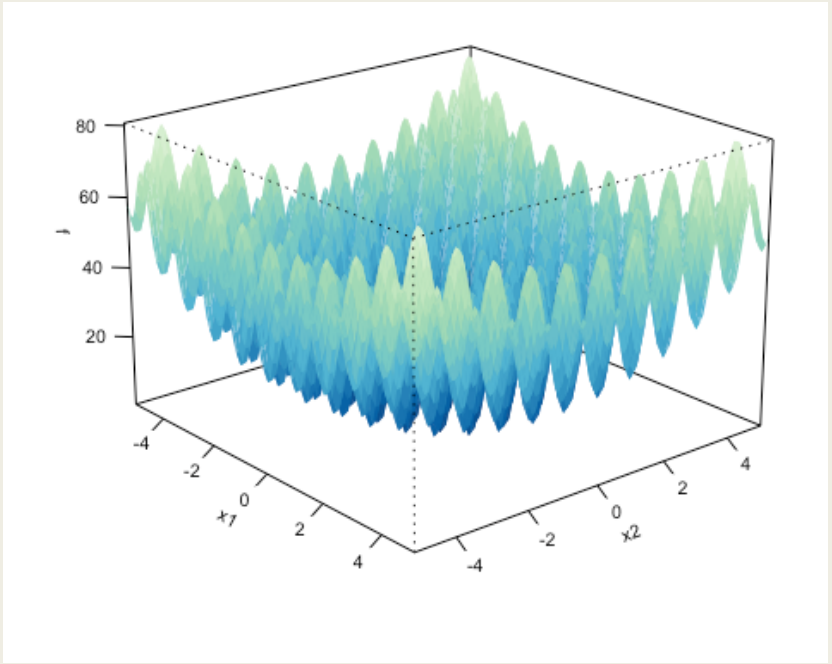
GA – Exercício 2

- Resolução

```
GA <- ga(type = "real-valued",
         fitness =  function(x) -Rastrigin(x[1], x[2]),
         min = c(-5.12, -5.12), max = c(5.12, 5.12),
         popSize = 50, maxiter = 1000, run = 100)
summary(GA)
```

GA – Exercício 3

- Encontrar o máximo da função
  - *f <- function(x)*
    *{ 100 * (x[1]^2 - x[2])^2 +*
    (1 - x[1])^2 }
- Com as restrições:
  - *c1 <- function(x)*
    *{ x[1]*x[2] + x[1] - x[2] + 1.5 }*
  - *c2 <- function(x)*
    *{ 10 - x[1]*x[2] }*



GA – Exercício 3

- Resolução

```
fitness <- function(x)
{
  f <- -f(x)                    # we need to maximise -f(x)
  pen <- sqrt(.Machine$double.xmax)  # penalty term
  penalty1 <- max(c1(x),0)*pen       # penalisation for 1st inequality constraint
  penalty2 <- max(c2(x),0)*pen       # penalisation for 2nd inequality constraint
  f - penalty1 - penalty2            # fitness function value
}
```

GA – Exercício 3

- Resolução

```
GA = ga("real-valued", fitness = fitness,
        min = c(0,0), max = c(1,13),
        maxiter = 5000, run = 1000, seed = 123)
summary(GA)
```

GA – Exercício 4

- Utilizando o problema anterior do Knapsack utilize o package GA para o resolver.
- Lembre-se que:
  - *Deve utilizar o modo binary e não real value*
  - *Use as funções auxiliares do package para a construção dos cromossomas*
  - *Leia a documentação do package*

ALGORITMOS GENÉTICOS EM R

Mestrado Integrado em Engenharia Informática
Mestrado em Engenharia Informática
Computação Natural

João Ramos
Fábio Silva
Paulo Novais