```r
#

# Script for Lecture 19 (Activity 1a, 1b, 1c)


######## Set the working environemnt

# Remove all variables from the R environment to create a fresh start
rm(list=ls())

# Set the working folder -- FILL IN THE LINE BELOW
setwd("~/Documents/SUTD/Term 5/ESA/Week 11/Lecture 19")


######## Packages we are going to use in this session: GA

# Package for Genetic Algorithms
if(!require(GA)){
  install.packages("GA")
  library(GA)
}


######## Warm-up Activity (Binary Genetic Algorithm)

# Define data
p <- c(6, 5, 8, 9, 6, 7, 3) # Value
w <- c(2, 3, 6, 7, 5, 9, 4) # Weights
W <- 9                      # Knapsack ’s capacity
n <- length(p)              # Number of items

# Define the fitness function. Note that we add a penalty to account for
the constraint on the Knapsack capacity.
# in this case, fitness is the sum of each decision variable (0 or 1) * the
corresponding value
# calculate the fitness given a value of the decision variable
knapsack <- function(x) {
  f <- sum(x * p)
  # abs portion is to account for the fact that the capacity of knapsack
may be violated
  penalty <- sum(w) * abs(sum(x * w) - W)
  f - penalty
}
```

```r
# Run SGA
SGA <- ga(type="binary",    # We use this option for binary decision
variables
          fitness=knapsack,  # We use the function'knapsack' to calculate
the fitness value
          nBits=n,           # Length of the bit string
          maxiter=100,       # The maximum number of iterations to run
before the GA search is halted.
          popSize=100)      # The population size

# best represents the best out of all finished iterations

# Use this function to analyze the results and the main settings of the GA
summary(SGA)
# elitism: number of solutions to keep as we move from one generation to
the next


# Graphical illustration of the search process of all iterations
plot(SGA)

# Solution
# to extract the best solution
x.star <- SGA@solution # Final solution: c(1, 0, 0, 1, 0, 0, 0)
sum(x.star)     # Total number of selected items: 2
sum(x.star * p) # Total profit of selected items: 15
sum(x.star * w) # Total weight of selected items: 9


######## Activity 1a -- Function optimization in 1D (real-valued)

# Define the function and the range of variability -- FILL IN THE LINE
BELOW
f <- function(x)  (cos(x) + x^3) * sin(x)
min <- 0; max <- 20
# Visualize the function
curve(f, min, max, n = 1000)

# Solve with GA and analyze the solution. Note that the GA function
maximizes by default!
# To solve the problem, we use 5 individuals and 5 generations -- FILL IN
THE LINE BELOW
GA_1a <- ga(type = "real-valued", fitness = function(x) -f(x), lower = min,
upper = max, popSize = 50, maxiter = 100)
```

```r
# after 10 iterations or so it converges (ie. optimised)
# Use this function to analyze the results and the main settings of the GA
summary(GA_1a)
# Graphical illustration of the search process
plot(GA_1a)

# Solution -- FILL IN THE LINE BELOW
GA_1a@solution # 17.4492
f(GA_1a@solution)        # -5236.023
# Note that the solution varies slightly if we re-run the GA. That's
because some steps of the search process are randomized.

# Graphically check the optimality of the solution
curve(f, min, max, n = 1000)
points(GA_1a@solution, -GA_1a@fitnessValue, col = 2, pch = 19)


######## Activity 1b -- Function optimization in 2D (real-valued)

# Define the function -- FILL IN THE LINE BELOW
# bivariate 2D function
Rastrigin <- function(x1, x2)
{ 20 + x1^2 + x2^2 - 10 * (cos(2*pi*x1) + cos(2*pi*x2)) }
# Visualize the function (3D view)
x1 <- x2 <- seq(-5.12, 5.12, by = 0.1)
f <- outer(x1, x2, Rastrigin)
persp3D(x1, x2, f, theta = 50, phi = 20, color.palette = bl2gr.colors)
# lots of local minima and maxima, hence finding an optimised value is not
trivial

# Visualize the function (2D view)
filled.contour(x1, x2, f, color.palette = bl2gr.colors)
# there is plenty of local minima and maxima

# Solve with GA and analyze the solution. Note that the GA function
maximizes by default! -- FILL IN THE LINE BELOW
GA_1b <- ga(type = "real-valued",
         fitness =  function(x) - Rastrigin(x[1], x[2]), # - in front of
Rastrigin because by default GA runs maximisation
         lower = c(-5.12, -5.12),
         upper = c(5.12, 5.12),
         popSize = 50,
         maxiter = 1000)   # Let's increase the number of generations

# Use this function to analyze the results and the main settings of the GA
```

```r
summary(GA_1b)
# Graphical illustration of the search process
plot(GA_1b)

# Solution -- FILL IN THE LINE BELOW
GA_1b@solution
# x1            x2
# -1.15633e-05 -2.980232e-05
Rastrigin(GA_1b@solution[,1], GA_1b@solution[,2])
# x1
# 2.027346e-07

# Note that the solution varies slightly if we re-run the GA. That's
because some steps of the search process are randomized.
GA_1b <- ga(type = "real-valued", fitness =  function(x) -Rastrigin(x[1],
x[2]),
            lower = c(-5.12, -5.12), upper = c(5.12, 5.12),
            popSize = 50, maxiter = 1000,
            seed = 10)

# Graphically check the optimality of the solution
filled.contour(x1, x2, f, color.palette = bl2gr.colors,
               plot.axes = { axis(1); axis(2);
                 points(GA_1b@solution[,1], GA_1b@solution[,2],
                        pch = 3, cex = 2, col = "white", lwd = 2) }
)


######## Activity 1c -- Constrained function optimization in 2D (real-
valued)

# Define the function and the constraints -- FILL IN THE LINE BELOW
f <- function(x)
{100 * (x[1]^2 - x[2])^2 + (1 - x[1])^2}
c1 <- function(x)
{ x[1]*x[2] + x[1] - x[2] + 1.5 }
c2 <- function(x)
{ 10 - x[1]*x[2] }

# Visualize the function and the feasible region
ngrid = 250
x1 = seq(0, 1, length = ngrid)
x2 = seq(0, 13, length = ngrid)
x12 = expand.grid(x1, x2)
col = adjustcolor(bl2gr.colors(4)[2:3], alpha = 0.2)
```

```r
plot(x1, x2, type = "n", xaxs = "i", yaxs = "i")
image(x1, x2, matrix(ifelse(apply(x12, 1, c1) <= 0, 0, NA), ngrid, ngrid),
      col = col[1], add = TRUE)
image(x1, x2, matrix(ifelse(apply(x12, 1, c2) <= 0, 0, NA), ngrid, ngrid),
      col = col[2], add = TRUE)
contour(x1, x2, matrix(apply(x12, 1, f), ngrid, ngrid),
        nlevels = 21, add = TRUE)

# Define a penalized fitness function to handle the constraints
fitness <- function(x)
{
  f <- -f(x)                         # we need to maximize -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
}

# Solve with GA and analyze the solution. Note that the GA function
maximizes by default! -- FILL IN THE LINE BELOW
GA_1c <- ga("real-valued", fitness = fitness,
        lower = c(0,0), upper = c(1,13),
        maxiter = 1000, seed = 10)

# Use this function to analyze the results and the main settings of the GA
summary(GA_1c)
# Graphical illustration of the search process
plot(GA_1c)

# Solution -- FILL IN THE LINE BELOW
GA_1c@solution
# x1       x2
# [1,] 0.8121198 12.31369

# wrong
# x1       x2
# [1,] 0.8025886 12.85142
# [2,] 0.8104237 12.35702
# [3,] 0.8085434 12.44749
# [4,] 0.7983696 12.86830
# [5,] 0.8110283 12.36115
# [6,] 0.7995823 12.84866
# [7,] 0.7973072 12.57544
# 100 * ...
f(c(0.8121198, 12.32369))
# 13605.28

# wrong
# 14901.78 13689.59 13909.27 14959.54 13696.94 14906.82 14255.78

# Graphically check the optimality of the solution
plot(x1, x2, type = "n", xaxs = "i", yaxs = "i")
image(x1, x2, matrix(ifelse(apply(x12, 1, c1) <= 0, 0, NA), ngrid, ngrid),
      col = col[1], add = TRUE)
image(x1, x2, matrix(ifelse(apply(x12, 1, c2) <= 0, 0, NA), ngrid, ngrid),
      col = col[2], add = TRUE)
contour(x1, x2, matrix(apply(x12, 1, f), ngrid, ngrid),
        nlevels = 21, add = TRUE)
points(GA_1c@solution[1], GA_1c@solution[2], col = "dodgerblue3", pch = 3)
```