```
using Images , StatsBase , Plots , PlutoUI , Distributions , Random
```

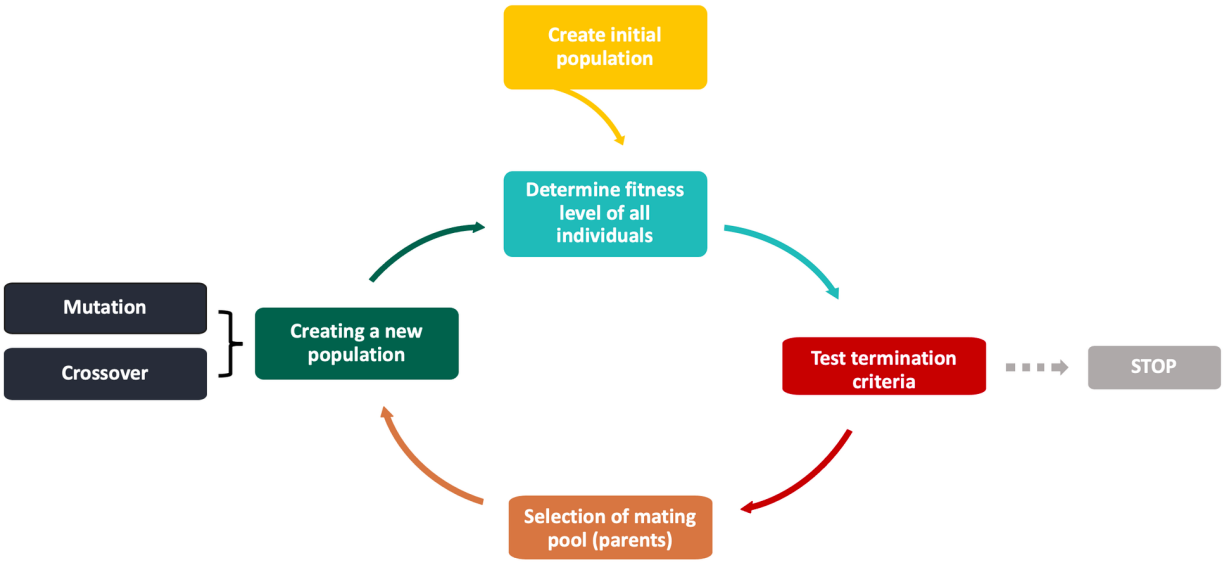
Genetic Algorithms

By Jordi Verbruggen & Jietse Verweirder

Introduction

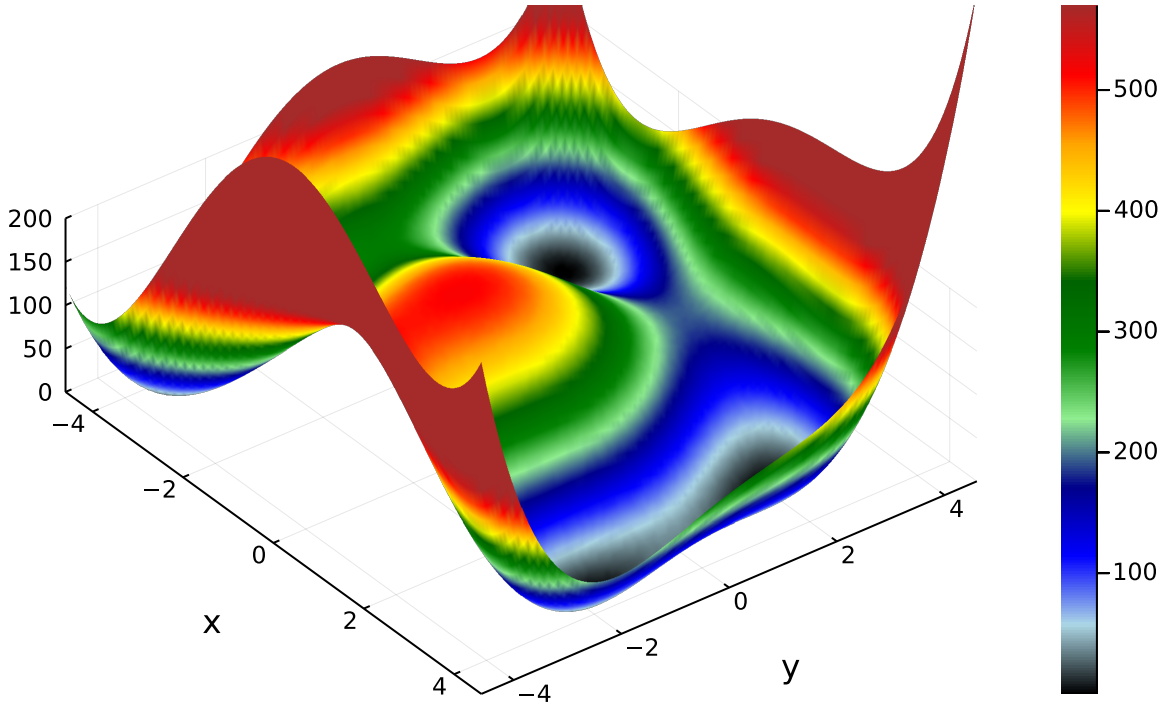
Genetic algorithms (GAs) are described as optimization algorithms that make use of concepts originated from biological evolution, such as natural selection and survival of the fittest. By introducing these concepts into the algotithm, a population will evolve to an optimal solution. **The population** is a representation of possible solutions, called chromosomes, to the problem in question. Each chromosome in their turn is comprised of genes, which are the actual number values of the solution. The algorithm will then apply evolutionary operators such as **mutation, recombination and selection**, over several generations, to converge the population to the best possible solution. **A fitness function** is used to determine which solution is the most optimal in the population, determining the objective function quality of the solution. Every generation, better solutions are created until **the termination criteria** are met or until a certain amount of generations has been run. The GA is terminated and the optimal solution should have been found. Although not guaranteed, most solutions will be of sufficient high quality.

A schematic overview of all the steps in the process is portrayed below.



$$f(x,y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$

A visualization of the function is given. *(Use the sliders to move the plot around.)*

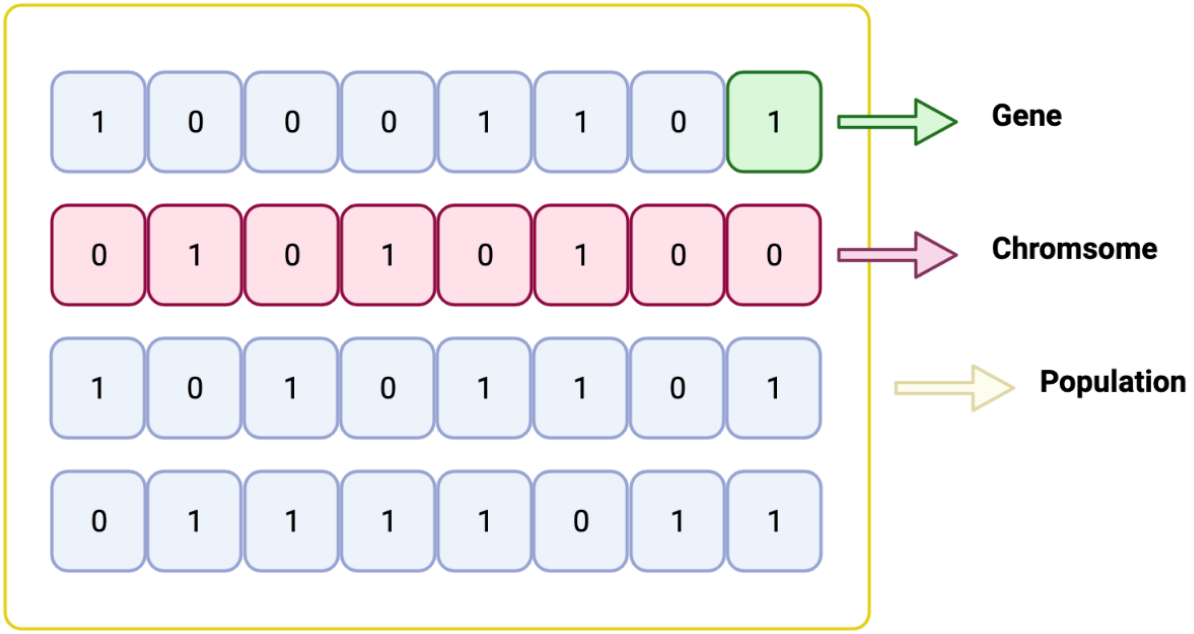


Angle 1: 50

Angle 2: 75

The function takes x and y coordinates that have real number values. Therefore the representation of the individuals will be a tuple of the x and y coordinates. In principle all techniques that are discussed here, are applicable to any function represented by chromosomes with real numbers. Bit string representations are also often used in GAs, and most of what is discussed further can be applied to that, but not all, which is important to keep in mind. To illustrate the implementation with bitstrings, figures of this process have been added.

In a last part of the notebook functionality has been added where custom objectives can be found via a genetic algorithm. Here you can write your function and by setting up some parameters, the extremeties of the objective can be found. **Important to mention here is that the functions have to be 2D or 3D.**



A few things have to be taken into account, when designing the initial population:

- **The population size:** A smaller population has a higher chance for a premature convergence, while a large population on the other hand asks for more computing time.
- **The diversity:** A low diverstity will also result in a premature convergence. Diversity in this context means that the individuals differ greatly in their genome.

Overall, there are two primary methods to initialize a population in a GA:

- **Random Initialization:** Populate the initial population with completely random solutions.
- **Heuristic initialization:** Populate the initial population using a known heuristic for the problem. The population exists of solutions that are already guided a few steps closer to the exact solution of the problem.

The code to initialize goes as folows:

```
"""
    random_init_pop(ps, gs, i_x, i_y)
```

```
        ]
    else
        pop = [[rand(Uniform(i_x[1], i_x[2]))] for _ in 1:ps]
    end

    return pop

end;
```

```
population =
 [[0.658548, 3.25798], [-2.22832, 0.781457], [-2.567, -1.52928], [1.63639, 3.73293], [-
```

```
population = random_init_pop(100, 2, [-4,4], [-4,4])
```

Fitness

As we want to improve our population over time, it is of absolute importance to select individuals that have a high **fitness level**. To determine the fitness level of an individual, we have to implement a **fitness function**. This function takes a candidate solution to the problem as input and produces a numeric value as output indicating how "fit" our how "good" the solution is.

Due to the fact that the fitness value has to be calculated after each generation, the fitness function should not only correlate closely with the designer's goal, but it also should be computationally efficient.

The fitness function used as example here is based on the himmelblau's optimization function:

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$

The x and y values respectively were the first and second gene in the chromosome of the individual.

```
    d = length(p[1])
    @assert(d == 1 || d == 2, "function has to be 2d or 3d")

    if d == 2
        if extremeties == "minima"
            pf = [[[ind[1], ind[2]], -ff(ind[1], ind[2])] for ind in p]
        else
            pf = [[[ind[1], ind[2]], ff(ind[1],ind[2])] for ind in p]
        end
    else
        if extremeties == "minima"
            pf = [[[ind[1]], -ff(ind[1])] for ind in p]
        else
            pf = [[[ind[1]], ff(ind[1])] for ind in p]
        end
    end

    return pf
end;
```

```
population_fitness =
 [[[0.658548, 3.25798], -71.67], [[-2.22832, 0.781457], -101.859], [[-2.567, -1.52928],
```

```
population_fitness = fitness(population, f_himmelblau, "minima")
```

Termination criteria

Eventually the run of the algorithm will have to come to an end. Therefore certain conditions

```
    return std([x[2] for x in pf]) < σ && nmax >= 5

end;
```

Selection

During selection, the parent chromosomes are choosen for later on "breeding purposes", this is implemented by **the crossover operator.** Different methods of selection are available and some are discussed below. Important to note here is that an individual can be selected more than one time, otherwise the population would not change. When each individual can only be selected once, the least fit individuals would also be selected for the mating pool. Therefore convergence of the algorithm would be hindered.

Roulette Wheel Selection

As the name suggests, this way of selecting parents is based on turning a roulette wheel. The chance of selecting an individual is proportional to its fitness. The higher its fitness the bigger its pocket will be on the roulette wheel and thus resulting in a higher chance of being selected.

```
    # choosen proportional to its fitness
    pool = sample(p, Weights(w), length(population_fitness))

    return pool

end;
```

```
pool_roulette =
 [[0.950116, -0.757216], [1.41526, -1.00724], [-0.566753, -0.277651], [-0.566753, -0.27
```

```
pool_roulette = roulette_wheel_selection(population_fitness)
```

Rank Selection

In **rank selection** individuals are ranked based on their fitness. The individual with the worst fitness is given rank one, the individual with the best fitness is given rank N. In rank selection, the

```
end;
```

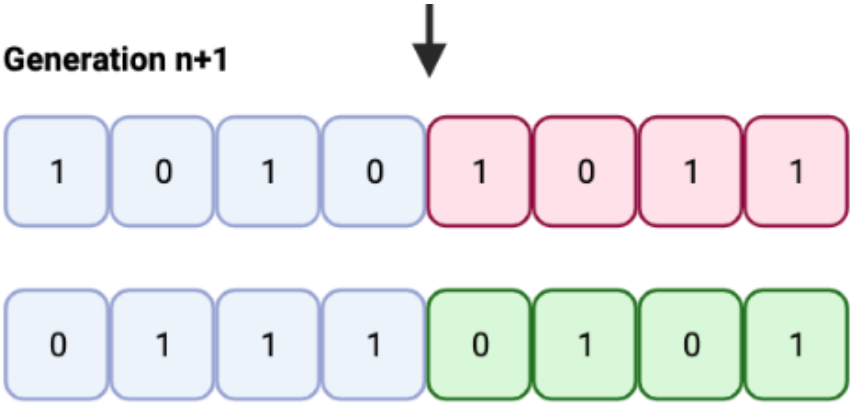
```
pool_rank =
  [[-1.21562, 2.56827], [-1.56061, 1.98179], [2.99085, 3.19307], [-3.03857, -0.900521],
```

```
pool_rank = rank_selection(population_fitness)
```

Steady State Selection

During **steady state selection** a proportion of individuals with the highest fitness is choosen as

Where:


Generation n+1
1 0 1 0 1 0 1 1
0 1 1 1 0 1 0 1