Readme

The Class\_Example.m script operates a Genetic Algorithm (GA) that evaluates the mathematical problem:

Subject to:

The GA contains the following functions:

|  |  |
| --- | --- |
| **Function** | **Description** |
| *InitialiseParameters* | Outputs a “struct” object with the GA parameters (including population size, mutation and crossover parameters among others). |
| *GeneratePopulation* | Creates an initial randomised solution pool to start the GA. |
| *GenerateChromosome* | Creates a single solution using uniform random distribution. |
| *FitnessMeasure* | Calculates for every solution in the population considering the problem constraints. |
| *SaveBestSolution* | Records the highest fitness solution found at every generation and over all generations. |
| *Selection* | Determines which solutions will be passed to the next generation. An elitist method is used where the best solutions are recorded, and the rest replaced by new solutions. |
| *Crossover* | Creates new “child” solutions by combining two “parent” solutions. A uniform crossover algorithm is employed, where each gene has a 50% chance to be passed on by one of the parents. The offspring will contain opposite genes (i.e. if child 1 has a gene from the parent 1, child 2 will contain a gene from the parent 2). |
| *Mutation* | Alters existing solutions. A uniform mutation process is used, so the gene to be mutated is selected at random, and the new gene value is also generated randomly. |

Two data types that may be unfamiliar are introduced:

|  |  |
| --- | --- |
| **Variable Type** | **Description** |
| *Cell* | Data type that may contain any type of data inside each container. It is indexed, and behaves similarly to any other array type. A new index notation is introduced: {row, column} allows the user to access the data **inside** the cell container. Using the notation (row, column) on a cell array will extract the cell container, rather than the information inside.  Example: *cellarray*{1,3}(2,1) will access the data indexed (2,1) in the array inside the cell {1,3} of our *cellarray*. On the other hand, *cellarray*(1,3) will output the cell container indexed at (1,3). |
| *Struct* | Allows the assignment of multiple variables to a single data structure. This is particularly useful to group variables of similar nature and pass multiple variables in a single input. *Structs* are introduced by using a period “.”.  Example: *structname*.*variable = 5* will create the *struct* “*structname*” and include 5 inside *variable*. The same notation is used to extract the information: i.e. *a = structname.variable* will yield *a = 5*. |